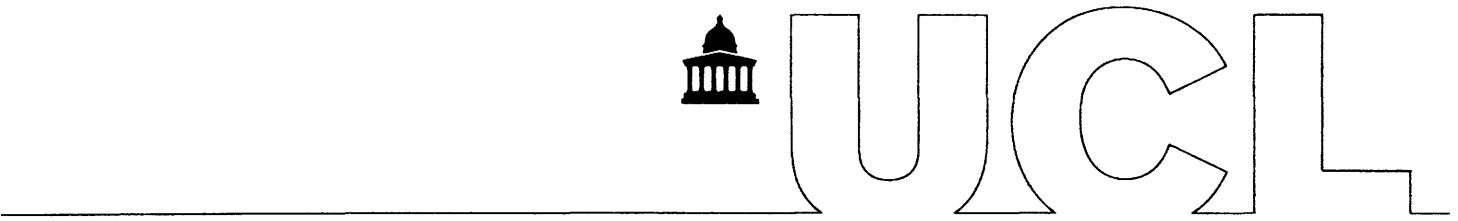
UNIVERSITY OF LONDON THESIS

Degree PhD Year 2008 Name of Author ELVAS DUARTE DE ALMEIDA, José-Paulo



A Graph-based Technique for Analysis and Visualisation of Higher Order Urban Topology

A thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
by

José-Paulo Elvas Duarte de Almeida

University College London
Department of Geomatic Engineering

2007

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

Analysis of spatial phenomena is a time consuming and laborious task in several fields of the Geomatics world. Automation of these tasks is especially needed in areas such as geographical information science (GISc). Carrying out those tasks in the context of an urban scene is particularly challenging given the complex spatial pattern of its elements.

The starting point in this study is unstructured data, and hence no prior knowledge of the spatial entities is assumed. The aim of translating this data into more meaningful homogeneous regions can be achieved by grouping geographic structures within the initial collection of objects according to their spatial arrangement.

The techniques applied to achieve this are those of graph theory, applied to urban topology analysis. For the identification of meaningful structures a graph-based system was developed comprising, in particular, a procedure, the *containment-first search*, based on the breadth-first search algorithm for graph traversal which, by considering the spatial objects' adjacencies, analyzes and interprets their spatial arrangement in terms of the topological relationship of containment. Different LiDAR as well as photogrammetric datasets have been used as an example scenario to test this system.

Another aspect is the visualisation of the urban topology. Visual inspections of interim results of the graph analysis process can often reveal patterns not discernable by current automated techniques. An interactive tool was developed and implemented in Visual Basic for Applications (VBA) utilising ESRI's ArcObjects, in ArcMap (ArcGIS). The ultimate goal was the dynamic display of the original map according to the results of the spatial topology analysis.

Future work will entail further clustering of the identified containment units into homogeneous regions. After the delineation of cluster shapes, an analysis process will have to be accomplished, either by pattern recognition or interpretation procedures, for the retrieval of higher-level information, for example related to land-us.

Acknowledgements

It is impossible to explicitly mention everyone who contributed, either directly or indirectly, to this work and made it possible. Even so, I would like to acknowledge the following persons.

First of all, my special thanks and deepest gratitude must go to my supervisors Mr. Jeremy Morley and Prof. Ian Dowman, for their permanent availability to guide me throughout this project from its very beginning. Their patience, inspiration, technical support and the fruitful discussions with them have been of extreme value.

I also want to express my deepest thanks to all other people in the UCL Department of Geomatic Engineering, including my peers, academic, research and support staff, for their help and for such a friendly environment provided. A special reference must be made to Dr. Muki Haklay and Dr. Rodérick Béra. With no shadow of a doubt, I cannot forget my colleagues Claire Ellul, Katerina Christopoulou and Samson Ayugi, for their companionship and for the fruitful brainstormings that we had so often. Another reference has to be made to the 2002/2003 MSc students, especially the GIS people with whom I started this long walk.

My gratitude is also due to Dr. João Coutinho-Rodrigues, my co-supervisor at the University of Coimbra, Portugal (Department of Civil Engineering, Faculty of Science and Technology), for his interest in this research and his constant encouragement. I also want to express my recognition to the academic and support staff of the Department of Mathematics and its Geomatic Engineering Group, for their help and encouragement to carry on, since the very beginning but especially during the last stage of this work.

Acknowledgements are also due to the institutions which have funded this work: *Faculdade de Ciências e Tecnologia da Universidade de Coimbra* (Faculty of Science and Technology of the University of Coimbra) and *Fundação para a Ciência e Tecnologia* (the Portuguese Foundation for Science and Technology; scholarship SFRH/BD/9909/2002).

For their love and for providing me such a cosy shelter to escape from town, a special reference is due to Angela and Nigel Stagg, they have been my adoptive family in England.

Last but not least is the profound recognition I want to express to all my family, especially my mother, my brother and sister, my nephew and niece, and also to Maria-Manuel not only for their love, unconditional enthusiasm and encouragement to go ahead, but fundamentally for their comprehension of my absence that this project has forced into. To you, all my love.

Thank you all truly much!

Contents

Abstract **3**
Acknowledgements **5**
Contents **7**
List of figures **11**
List of tables **17**
Introduction **18**
1.1 General context 18
1.2 Thesis outline 20
2 GIS and topology **23**
2.1 Preliminaries 24
2.2 Mathematical topology 26
2.2.1 Pointset topology 27
2.2.2 Algebraic topology 29
2.3 Topological frameworks 32
2.4 Networks and areas *vs.* graphs 37
2.4.1 Topological consistency 37
2.4.2 Planar enforcement 40
2.5 Digital cartography 42
2.6 The ESRI's coverage data model 45
2.7 Summary and discussion 48
3 Graph theory **50**
3.1 Generalities 50
3.2 Basic principles and definitions 51
3.3 Types of graphs 55
3.4 Graph search: depth-first search *vs.* breadth-first search 59
3.4.1 Depth-first search 60
3.4.2 Breadth-first search 61
3.5 How graph theory is applied to spatial analysis 63
3.5.1 The eXtended Relational Attributed Graph (XRAG) 63
3.5.1.1 Introduction 63
3.5.1.2 Description 63
3.5.2 Space syntax 67
3.5.3 β-Skeletons 75
3.5.3.1 Introduction 75
3.5.3.2 Description 77

3.5.3.3 Examples of applications 78
3.5.4 Urban network analysis 79
3.5.5 Built-form connectivity 88
3.6 Summary 95
4 Thesis objectives and research process 97
4.1 The need for more investigation 97
4.2 Thesis aims 98
4.3 Research methodology 99
4.4 Summary 101
5 Test data 102
5.1 LiDAR system, data and applications 103
5.1.1 General principles 103
5.1.2 Applications of LiDAR data 107
5.2 Description of the LiDAR datasets 110
5.2.1 The London dataset 110
5.2.2 The Stuttgart datasets 112
5.3 Description of the photogrammetric datasets 114
5.3.1 The Barton-Bendish dataset 115
5.3.2 The Heerbrugg dataset 117
5.3.3 The Santa Lucija dataset 119
5.4 Pre-processing 120
5.4.1 Internal description of the point patterns 121
5.4.2 Preparation of the polygon data base 123
5.5 Summary 128
6 A containment-first search algorithm 129
6.1 A network of connectivity across the polygonal regions 131
6.2 Construction of graphs 132
6.2.1 Retrieval of the polygon adjacencies 132
6.2.2 Retrieval of the steep-polygon touchings 136
6.2.3 Graph data structures 139
6.2.3.1 Adjacency-matrix *vs.* adjacency-list representations 139
6.2.3.2 The adjacency-list representation of the graphs obtained 142
6.3 The method for spatial topology analysis 146
6.3.1 Preliminaries 146
6.3.2 Containment-first search 149
6.3.3 Rationalization 151
6.3.4 Colour selection 153
6.3.5 Verifying the algorithm for spatial topology analysis 154
6.3.6 Overcoming limitations 157
6.3.6.1 The problem 157
6.3.6.2 Polygon-ring containment 159
6.3.7 Coding the algorithm 164
6.3.7.1 The main program 165
6.3.7.2 Dissection of process (A): adjacencies graph traversal 167
6.3.7.3 Dissection of process (B): adjacency graph cycles detection 169
6.3.7.4 Dissection of process (C): containment-first search and rationalization 170
6.4 Visualisation of the analyses 174

6.4.1 Designing the interactive tool 175
6.4.2 Coding the interactive tool 176
6.5 Summary 181
7 Proof of concept 184
7.1 Generation of synthetic data 184
7.2 Construction of graphs 187
7.3 Spatial topology analysis 189
7.4 Summary 192
8 Validation and discussion 193
8.1 Experiments with LiDAR data 194
8.1.1 Raw data 194
8.1.1.1 The London dataset 194
8.1.1.2 The Stuttgart datasets 199
8.1.2 Morphologically filtered data 207
8.1.2.1 Generalities about morphological filtering 207
8.1.2.2 The Stuttgart datasets 208
8.2 Experiments with photogrammetric data 232
8.2.1 The Barton-Bendish dataset 232
8.2.2 The Heerbrugg dataset 237
8.2.3 The Santa Lucija dataset 240
8.3 Statistical evaluation of key results 245
8.3.1 Quantitative assessment method 245
8.3.2 Reference data for quality assessment 248
8.3.3 Quantitative assessment results (reference data: aerial image) 255
8.3.4 Quantitative assessment results (reference data: LiDAR data) 261
8.4 Summary and discussion 267
9 Conclusions 270
9.1 General discussion 270
9.1.1 Purpose of the study 270
9.1.2 Aims and objectives 271
9.1.3 Research outcome 272
9.1.4 Limitations 272
9.1.5 Value and possible applications 273
9.2 Recommendations and future research 274
References 276

Appendix A AML routines to retrieve polygon adjacencies 285
Appendix B Extract from a file of a graph of adjacencies 288
Appendix C AML routine to retrieve steep-polygon touchings 290
Appendix D Example of a file of a graph of steep-polygon touchings 296
Appendix E The spatial topology analysis program 298
Appendix F Extract from a file of a graph spanning tree 314
Appendix G Extract from a file of graph cycles 316
Appendix H Extract from a CFS and Ratinalization output file 318
Appendix I Visualisation of the spatial topology analyses: the interactive tool code 320

Appendix J Paper in press for publication **329**
Appendix K CD ROM **346**
Appendix L Other types of graphs **347**
Common families 347
Graph modelling applications 348

List of figures

Figure 2.1 – Points and their neighbourhoods in a topological space. 27

Figure 2.2 - The London Underground map embedded in a geographical map. 29

Figure 2.3 – The London Underground map - a topological transformation 29

Figure 2.4 – Examples of 0-, 1- and 2-simplices. 30

Figure 2.5 - a) a simple line and b) a complex line; 32

Figure 2.6 – The 9-Intersection Model: a geometric interpretation of the eight topological relations between two spatial regions with connected boundaries and without holes. 34

Figure 2.7 – Illustrations of the eight RCC8 relations. 35

Figure 2.8 – The Calculus-based Method: topological situations illustrating the 36

Figure 2.9 - The Calculus-based Method: topological situations illustrating the 36

Figure 2.10 – Maps and respective graphs; the USA Census Bureau topological cells. 38

Figure 2.11 – The planar enforcement process. 40

Figure 2.12 – Additional complexity added by planar enforcement. 41

Figure 2.13 – Examples of routes and sections in the ESRI's coverage data model: a) route starting and ending at an arc end; b) route starting at a point along an arc. 42

Figure 2.14 – Examples of regions in the ESRI's coverage data model. 42

Figure 2.15 – Some conditions for digitized maps. 43

Figure 2.16 – The ArcInfo coverage data model for storing vector data. 47

Figure 3.1- An example for part of a street map. 52

Figure 3.2 - A diagram representing the street map sample above. 52

Figure 3.3 - Another possible representation of the street map sample in Figure 3.2. 53

Figure 3.4 - A different aspect of the same graph in Figure 3.2 and Figure 3.3 53

Figure 3.5 - A general example of dual graphs. 58

Figure 3.6 - Example of a map of a simulated scene and respective graph of adjacencies. 60

Figure 3.7 - Example of two different traversal trees for the graph in Figure 3.6, ... 60

Figure 3.8 – The inference process of second-order thematic information about a scene 63

Figure 3.9 – **a)** A simple urban scene (from Barnsley, 2003); **b)** Graph representation of the spatial relation of ADJACENCY between the land-cover parcels (from Barnsley, 2003); **c)** Graph representation of the spatial relation of CONTAINMENT between the land-cover parcels (after Barnsley, 2003). 64

Figure 3.10 – XRAG data structure. 66

Figure 3.11 – a) Mapping a schematic setting onto a graph; b) mapping non-convex spaces (vertices may not capture the actual configuration of the spatial structure of a setting). 69

Figure 3.12 – Schematic structure of Axwoman. 74

Figure 3.13 – As an example, two different α-hulls of a common point set: 75

Figure 3.14 – A β-skeleton of the point set showed in Figure 3.13. 76

Figure 3.15 – Route-solutions (represented by points) in a two-objective space. 80

Figure 3.16 – Graphical representation of solutions (for illustrative purposes, solution 7) 83

Figure 3.17 – Solution basket window with columns graph and value paths display. 84

Figure 3.18 – Search settings window with additional constraint. Solution 7 has been directed to be shown; this solution had constraints on Objectives 1 and 5, and equal weights on the objectives. 85

Figure 3.19 – Facility location and building assignment for p=4 (Solution #8, Goal L1). 86

Figure 3.20 – Example of Primary and Secondary paths for two given buildings. 87

Figure 3.21 – Conceptual hierarchical levels of resolution of Krüger's urban graph system. 90

Figure 3.22 – The five universes U_n, n = 1,…,5, used to describe the structure of built forms and their urban settings: U_1 describes the partition (party-wall) condition; U_2 describes the external-wall condition; U_3 describes the connections of built forms to the street system; U_4 describes the street network; and U_5 describes the relationship between city blocks sharing the same streets. 91

Figure 4.1– An overview of the research methodology. 100

Figure 5.1 - Principle of LiDAR, or Airborne Laser Scanning. 104

Figure 5.2 - LiDAR dataset used for development purposes of the graph-based technique – Kew, southwest London. Data is colour coded in elevation range. 111

Figure 5.3 – The Stuttgart's Site 1 cloud of LiDAR points. 113

Figure 5.4 - The Stuttgart's Site 2 cloud of LiDAR points. 114

Figure 5.5 –The area of interest in Barton-Bendish. 116

Figure 5.6 – The photogrammetric 3D point cloud of Barton-Bendish. 117

Figure 5.7 - The photogrammetric 3D point cloud of Heerbrugg. 118

Figure 5.8 - The photogrammetric 3D point cloud of Santa Lucija. 119

Figure 5.9 – Overview of the main steps of the pre-processing of unstructured data. 121

Figure 5.10 - DSM generated from the LiDAR point set of Kew - southwest London. 123

Figure 5.11 –The TIN facets binary classification thresholding. 124

Figure 5.12 – Binary classification of TIN facets – Kew, southwest London 125

Figure 5.13 – A detail of the southwest side of the surveyed area depicted in Figure 5.12b). 125

Figure 5.14 – A PAT info file showing the binary classification of the TIN facets: "flat" polygons (SLOPECLASS = 0), and "steep" polygons (SLOPECLASS = 1). 127

Figure 6.1 – An overview of the main steps of the graph-theoretic algorithm 130

Figure 6.2 – Establishing a network of connectivity between polygonal regions. 131

Figure 6.3 – Graph of adjacencies for the London dataset's case study area (generated using Ucinet 6 for Windows; Borgatti et al., 2002); boxes refer to the areas enlarged in Figure 6.9. 134

Figure 6.4 – Graph of touchings between steep polygons for the London dataset's case study area (generated using Ucinet 6 for Windows; Borgatti et al., 2002). 139

Figure 6.5 – Two different adjacency-list representations of the simulated graph in Figure 3.6: 141

Figure 6.6 –Spanning trees for the simulated graph in Figure 3.6: a) DFS tree – graph stored using adjacency-list representation with linked lists sorted by vertex index ascending; b) BFS tree - vertex index ascending; c) DFS tree – vertex index descending; d) BFS tree - vertex index descending. 142

Figure 6.7 – The adjacency-list representation of a graph within a computer. 143

Figure 6.8 – Adding edges to a graph represented in a computer by adjacency lists: 144

Figure 6.9 – Detail of Figure 6.3: the interpretation of the path highlighted is an example of the geographical information that can be inferred from a graph path, in the context of an urban scene (the numbers on the map are polygon labels in insert). 148

Figure 6.10 – TIN generated for the Kew area (London dataset); the rectangle in yellow highlights the building shaped by the rectangular dark green polygon in Figure 6.9 148

Figure 6.11 – CFS and the detection of potential spatial features: each vertex adjacent to the root ... 151

Figure 6.12 – An example of the generalisation procedure: the steep-polygon islands, mostly related to noise, are ignored. 151

Figure 6.13 – The process of "clearing out" steep-polygon islands within flat polygons. 152

Figure 6.14 – The HSV colour system in ArcMap 153

Figure 6.15 – First experiment of the implemented methodology, ... 155

Figure 6.16 – Detail of the southwest part of the map pictured in Figure 6.15. 156

Figure 6.17 – The graph search paths followed by the CFS algorithm 157

Figure 6.18 - Detail of Figure 6.9: a special case of the spatial relation containment, ... 158

Figure 6.19 – Detection of polygon-ring containments using graph cycles, showing edge effects. ... 160

Figure 6.20 - The containment-first search process: 162

Figure 6.21 – A second experiment carried out with the improved CFS, extended to detect polygon-ring containments. Experiment applied to the case study area of the London dataset 163

Figure 6.22– Overview of the spatial topology analysis program 165

Figure 6.23 – The process of traversing the graph of adjacencies by using either the DFS or BFS algorithm; generation of the respective spanning tree. 167

Figure 6.24 – Detection of cycles in the graph of adjacencies. 169

Figure 6.25 – The CFS & Rationalization process 171

Figure 6.26 – A prototype of an interactive application for the visualisation of spatial topology. 175

Figure 6.27 – Starting the interactive application: selection of features on the steep/flat region map. 176

Figure 6.28 – Form 1 object - Spatial Topology Analysis (vd. Figure 6.26); 177

Figure 6.29 – Construction and display of the spanning tree of the adjacency graph; dynamic mapping of the slope regions in the original map according to the CFS and Rationalization results 178

Figure 6.30 – Part of a PAT info file after the results of CFS and Rationalization 180

Figure 6.31 - Summary of the analysis process of urban topology, ... 183

Figure 7.1 – Building footprints taken from OS Master Map data[10]: 185

Figure 7.2 – Simulated map of gradient regions: binary classification of the polygons generated into 186

Figure 7.3 – Polygon attribute table (PAT) of the simulated map of gradient regions 187

Figure 7.4 – Graph of adjacencies of the simulated map of gradient regions. 187

Figure 7.5 – Spatial topology analysis and the different containment units identified. 189

Figure 7.6 – Branches of the breadth-first search tree directly relate to spatial features 190

Figure 7.7 – Sequences of containments correctly detected within a given containment unit. 191

Figure 7.8 – Polygon attribute table (PAT) of the simulated map of gradient regions 191

Figure 8.1 –The urban topology analysis and the different containment units identified – 195

Figure 8.2 – The binary classification of the TIN facets and the generation of steep/flat polygons – 197

Figure 8.3 - The urban topology analysis and the different containment units identified – 198

Figure 8.4 – The DSM generated from the LiDAR point set of Stuttgart – Site 1 (original data). 199

Figure 8.5 – The DSM generated from the LiDAR point set of Stuttgart – Site 2 (original data). 200

Figure 8.6 – The binary classification of the TIN facets and the generation of steep/flat polygons – 201

Figure 8.7 – The binary classification of the TIN facets and the generation of steep/flat polygons – 202

Figure 8.8 - The urban topology analysis and the different containment units identified – 204

Figure 8.9 - The urban topology analysis and the different containment units identified – 206

Figure 8.10 – The DSM generated from the morphologically filtered LiDAR data - Stuttgart, Site 1. 208

Figure 8.11 – The DSM generated from the morphologically filtered LiDAR data – Stuttgart, Site 2. 209

Figure 8.12 – The binary classification of the TIN facets and the generation of steep/flat polygons – 210

Figure 8.13 – The binary classification of the TIN facets and the generation of steep/flat polygons – 211

Figure 8.14 - The urban topology analysis and the different containment units identified – Crosses indicate existence of sliver polygons; circles indicate urban features not identified as containment units. 213

Figure 8.15 - The urban topology analysis and the different containment units identified – Crosses indicate existence of sliver polygons; circles indicate urban features not identified as containment units. 214

Figure 8.16 – The binary classification of the TIN facets and the generation of steep/flat polygons - Morphologically filtered data; 60° gradient thresholding. The yellow box represents a sub-dataset....216

Figure 8.17 – The binary classification of the TIN facets and the generation of steep/flat polygons – 217

Figure 8.18 - The urban topology analysis and the different containment units identified –....219

Figure 8.19 - The urban topology analysis and the different containment units identified –....220

Figure 8.20 – The DSM generated from a sub-dataset of the 221

Figure 8.21 – The binary classification of the TIN facets and the generation of steep/flat polygons – 222

Figure 8.22 - The urban topology analysis and the different containment units identified–....222

Figure 8.23 – The binary classification of the TIN facets and the generation of steep/flat polygons – 225

Figure 8.24 – The binary classification of the TIN facets and the generation of steep/flat polygons – 226

Figure 8.25 - The urban topology analysis and the different containment units identified –....227

Figure 8.26 - The urban topology analysis and the different containment units identified –....228

Figure 8.27 – The urban topology analysis: the identified containment units 230

Figure 8.28 – The urban topology analysis: the identified containment units 231

Figure 8.29 – The DSM generated from the photogrammetric point set of Barton-Bendish. 232

Figure 8.30 – The binary classification of the TIN facets and the generation of steep/flat polygons - 233

Figure 8.31 - The urban topology analysis and the containment units identified –....234

Figure 8.32 - The binary classification of the TIN facets and the generation of steep/flat polygons -236

Figure 8.33 - The urban topology analysis and the containment units identified –....237

Figure 8.34 - The DSM generated from the photogrammetric point set of Heerbrugg. 238

Figure 8.35 – Extract from the Heerbrugg photogrammetric data: 239

Figure 8.36 - The DSM generated from the photogrammetric point set of Santa Lucija. 240

Figure 8.37 - The binary classification of the TIN facets and the generation of steep/flat polygons -241

Figure 8.38 - The urban topology analysis and the containment units identified –....242

Figure 8.39 - The binary classification of the TIN facets and the generation of steep/flat polygons -243

Figure 8.40 - The urban topology analysis and the different containment units identified – 244

Figure 8.41 – Illustration of the computation of Shufelt metrics. 247

Figure 8.42 – The reference scene model for Site1 of Stuttgart, ...250

Figure 8.43 - The reference scene model for Site2 of Stuttgart, ...251

Figure 8.44 - The reference scene model for Site1 of Stuttgart, ...253

Figure 8.45 - The reference scene model for Site2 of Stuttgart, ...254

Figure 8.46 – Assessment of the containment unit classification for Site 1 - reference data generated from the aerial image: (A **and** B) polygons represent areas where reference and classification coincide; (**not** A **and** B) are classified areas which are not part of the reference areas; (A **and not** B) are reference areas which are not classified. 256

Figure 8.47 – Assessment of the containment unit classification for Site 2. 259

Figure 8.48 – Assessment of the containment unit classification for Site 1. 262

Figure 8.49 – Assessment of the containment unit classification for Site 2. 265

Figure L.1 – A general example of a multi-graph. 349

Figure L.2 – A general example of a digraph. 350

Figure L.3 - A general example of a random graph. 351

Figure L.4 - A general example of a hypergraph. 352

List of tables

Table 3.1 – Krüger's (1979a, 1979b) graph-theoretic measurements of built-form connectivity (V ≡ number of vertices; E ≡ number of edges; C ≡ number of components; n refers to the n^{th} graph universe, n=1,...5).....93

Table 8.1 – Computation of Shufelt metrics.....247

Table 8.2 – General statistics for reference and classified polygons in Site 1.....257

Table 8.3 – Shufelt's producer and user accuracies for the containment unit classification in Site 1.257

Table 8.4 – General statistics for reference and classified polygons in Site 2.....260

Table 8.5 – Shufelt's producer and user accuracies for the containment unit classification in Site 2.260

Table 8.6 – General statistics for reference and classified polygons in Site 1.....263

Table 8.7 - Shufelt's producer and user accuracies for the containment unit classification in Site 1. 263

Table 8.8 – General statistics for reference and classified polygons in Site 2.....266

Table 8.9 – Shufelt's producer and user accuracies for the containment unit classification in Site 2.266

Table 8.10 – Combined Shufelt's metrics for containment-unit classification for both Site 1 and Site 2. 268

1 Introduction

1.1 General context

Interpretation and analysis of spatial phenomena is a highly time consuming and laborious task in several fields of the Geomatics world. This is particularly evident given the fact that the new technologies continuously being developed to acquire spatial information, are yielding more accurate but also increasingly large and complex spatial datasets. In addition, the necessity of keeping these datasets up to date, and also given the different levels of detail of information required by applications with different purposes, justify the growing demand for the automation, or semi-automation, of those tasks in areas like Geographical Information Science (GISc), or digital cartography (Anders *et al.*, 1999).

If the initial dataset happens to comprise unstructured information, or information not explicitly structured for the required purposes, the tasks of interpreting and analysing spatial phenomena are even more challenging. The aim of retrieving structured information translated into more meaningful homogeneous regions can be achieved by: identifying meaningful structures within the initial random collection of objects; and by understanding their spatial arrangement (Barr and Barnsley, 1997; Anders *et al.*, 1999; Kim and Muller, 1999; Bauer and Steinnocher, 2001; Nardinocchi *et. al.*, 2003; Forlani *et. al.*, 2006).

The main idea is to find *higher-level structures* by delineating coherent, homogeneous structures in an arbitrary collection of lower-level details; a *higher-level structure* is defined within this thesis as a set of different buildings which may well include vegetation. For the purpose stated above, trying to find inherent, relevant relations in the initial unstructured data, and infer facts from that, is

necessary. In these circumstances, not only the attributes of an object are relevant, but most importantly its spatial location and how the object is spatially related to the other objects (Barr and Barnsley, 1997; Anders *et al.*, 1999; Kim and Muller, 1999; Bauer and Steinnocher, 2001; Nardinocchi *et. al.*, 2003; Forlani *et. al.*, 2006). This is actually the main goal of Data Mining, which emerged as an independent branch of computer science (Anders *et al.*, 1999).

An interesting aspect worth to explore is the spatial topological representation of the initially unstructured datasets. Indeed, the human brain is sensitive to the visual representation of real scenes, and visual analysis can often reveal patterns not discernable by current automated analysis techniques. Overall, structured spatial analysis fail to emphasize the power of the human eye in detecting patterns, and the role of subjectivity in GISc (Schuurman, 2004; Dykes *et al.*, 2005). Structured inquires are certainly intrinsic to a geographic information system (GIS), but "visuality and intuition" should be viewed as a distinct advantage to convey patterns and concepts. According to this perspective, GIS extended the capabilities of the analytical techniques by allowing the visualisation of spatial arrangements and, in the process, restored intuition as a valid heuristic technique (Schuurman, 2004; Longley *et al.*, 2005).

Intuition, which is related to cognitive processes, is used by GISc researchers as a means of interpreting visual display of geographical data. Links between visual cognition, knowledge discovery and information technology have been object of intense research during the past decade partly because visualisation represents a different approach from traditional decision-making processes; it is the relation of graphical display to communication of information that distinguishes the methodology (Schuurman, 2004).

Exploratory data analysis (EDA) and knowledge discovery in databases (KDD) are related methodologies that rely upon assumptions of direct correlation between visualisation and communication of information. Often referred to as Data Mining, KDD incorporates machine-learning computational techniques with visualisation in order to perceive patterns and to judge the suitability of data according to established criteria. KDD and EDA are iterations of what is known as *scientific visualisation*: a

combination of simulation, data analysis, and visualisation of complex scientific relationships (Schuurman, 2004).

Another important aspect that should be considered is the spatial context where the interpretation and analysis of the spatial phenomena take place. A city is of particular relevance not only because it is an important place for human livings, but it is also the centre of economy and social development (Zeng *et al.*, 2002). Accomplishing the tasks mentioned above in the context of an urban scene is particularly awkward given the huge number of relatively small component elements (such as buildings, trees, intra-urban open spaces, etc.) and their generally complex spatial pattern.

In particular, our ultimate interest is on the land-use classification of urban environments for it is a key role in several fields, such as urban land-use mapping, urban planning and management, establishment and revision of a GIS database, environment and disaster monitoring, etc. (Zeng *et al.*, 2002).

However, deriving information related to land-use is not a straightforward task. According to some authors (including Eyton, 1993, and Barnsley & Barr, 1996, both cited in Barnsley and Barr, 1997), the classification process of spatial information to produce land-cover maps (maps of *forms*) for urban areas can be considered fairly simple if we compare it with the process of deriving information from those maps on urban land-use (maps of *functions*), which is normally much more problematic. This is fundamentally due to the fact that "land-use is an abstract concept - an amalgam of economic, social and cultural facts – one that is defined in terms of *function* rather than physical *form*" (Barnsley and Barr, 1998; Bauer and Steinnocher, 2001; Barr and Barnsley, 2004). Thus, the relationship between land-use in urban areas and the spatial data acquired in surveys of those areas is very complex and indirect.

1.2 Thesis outline

In order to provide the reader a frame of reference, this chapter has given a brief description of the context of this thesis. The background of this research is given in more detail in the next two chapters, Chapter 2 and Chapter 3.

In Chapter 2 the fundamental concepts of topology, and how it emerged as one of the central defining features of a GIS, are presented. Concepts of mathematical topology with regards to homeomorphisms, and the properties that are preserved through homeomorphisms, are reviewed. The two main components of mathematical topology - pointset topology and algebraic topology - are also discussed. Some of the most commonly used topological frameworks in GIS are reviewed. This chapter also shows how the space can be treated as a combination of line segments, *i.e.* a network which in turn can be viewed as a graph. The geometric and principally the topological conditions for spatial data integrity in digital maps are reviewed. Because the ESRI's ArcInfo coverage data model was used in this research, the topological data structure implemented by the coverage data model is also described.

The general concepts of graph theory are reviewed in Chapter 3. After briefly introducing this subject, the basic principles of graph theory and its terminology are summarised. Only the concepts and types of graphs relevant to this research are included; other definitions, common graph families, and particular graphs to model specific situations are included in Appendix L. This chapter also overviews graph-search algorithms used in the process of learning structural properties of graphs. In a second part of this chapter, five examples of graph-based applications for spatial system analysis are reviewed.

Chapter 4 indicates the research questions of this thesis, its objectives and presents an overview of the methodology proposed.

Chapter 5 focuses on the test environment of this research by presenting the different types of real world data used. As an example scenario, *LiDAR* (*Li*ght *D*etection *a*nd *R*anging) data were used to develop and test the graph-based technique conceived in this work. The principles of LiDAR systems, the data and their main applications are briefly reviewed. The LiDAR datasets used for development and test purposes are described. In order to analyse the versatility of the developed algorithm, this was also applied to photogrammetric data; the datasets used are presented in this chapter. Finally, the steps undertaken in the preparation of the raw data are explained and summarised both in the form of flowcharts and pseudo code.

The methodology proposed and the technical aspects of its implementation are described in detail in Chapter 6. The conceptual considerations of how graph theory was applied in the design of a tool for the analysis and visualisation of spatial topology are presented. Details of the construction of the network of connectivity, throughout the pre-processed collection of spatial objects, are given. The foundations of the analytical analysis method are described. The first experiments carried out with the case study area of the London dataset are also described. The limitations of the technique that became evident from the initial experiments are indicated, and the process undertaken to extend the *containment-first search* procedure in order to address the limitations is explained. This chapter is concluded with the description of the implementation of a prototype of an interactive tool for the visualisation of the urban spatial topology.

Proof of concept is the object of Chapter 7. Before tests with real world data are presented, processing with synthetic data was undertaken in order to explicitly show that, in the absence of noise and error, the algorithms do indeed make explicit the urban spatial topology, and in particular that containment relationships do relate to *higher-level* urban information.

In order to test the algorithm's performance with real world data, experiments with LiDAR data and photogrammetric data were carried out; Chapter 8 describes and discusses the results obtained. The description of a statistical evaluation of key results is also given in this chapter.

Chapter 9 summarises the work presented in the thesis. Conclusions are drawn and an outlook on the future work is given.

A list of the bibliographical references is included. A series of appendices is also included, namely: extracts from different input and output files, programming code, a paper in press accepted for publication, and a CD ROM with the application implemented.

2 GIS and topology

In this chapter relationships between places/objects within the space are reviewed and examined from the perspective of relative placement – for instance, given two objects A and B, is object A next to object B? If so, do they share any boundary? Moreover, are they juxtaposed to each other, or is one of them completely surrounded by the other? These are the sorts of questions that the study of topology in a geographic information system (GIS) environment can answer.

This chapter is structured as follows. In section 2.1 the fundamental concepts of topology, and how it emerged as one of the central defining features of a GIS, are presented. Concepts of mathematical topology with regards to homeomorphisms, and the properties that are preserved through homeomorphisms, are reviewed in section 2.2; the two main components of mathematical topology, pointset topology and algebraic topology, are also discussed in sections 2.2.1 and 2.2.2 respectively. Section 2.3 reviews some of the most commonly used topological frameworks in GIS. Section 2.4 shows how the space can be treated as a combination of line segments, *i.e.* a network which in turn can be viewed as a graph; but, the representation of areas and networks as graphs requires both the topological consistency of the data model, and the planarity of the objects to be represented - these two aspects are discussed in sections 2.4.1 and 2.4.2 respectively. The geometric and principally the topological conditions for spatial data integrity in digital maps are reviewed in section 2.5. Because the ESRI's ArcInfo coverage data model was used for our purposes, section 2.6 describes the topological data structure implemented by the coverage data model. Finally, a summary and discussion of this chapter is made in section 2.7.

2.1 Preliminaries

One of the very first uses of topology as a problem-solving tool has been attributed to the Swiss mathematician Leonard Euler (generally recognised to be the founder of the systematic study of topology), who proved in 1736 that the famous problem of his time, called the Königsberg Bridge Problem, was unsolvable. In the same year, he published what is often referred to as the first paper in graph theory (Gibbons, 1989; Bollobás, 1998; Worboys 1995, 2004). Since then, topology has become a branch of mathematics, and more recently evolved as an area of functionality offered in GIS.

At this stage, the definition of a GIS, the principal environment of the work described in this thesis, should be recalled. Several definitions of a GIS from different authors can be found in the literature, though all agreeing on the main points. For the purposes of this document, we consider the definition given by Michael Worboys (1995, 2004):

> "A geographic information system (GIS) is a special type of computer-based information system that enables capture, modelling, storage, retrieval, sharing, manipulation, analysis, and presentation of geographically referenced data."

The definitive moment for the use of topology in GIS was the Advanced Symposium on Topological Data Structures in 1977, where topology, its place in GIS, and topology research and algorithms were discussed. The symposium validated topology's use and, beginning in the early 1980s, most GIS procurements mandated that spatial data be stored in topological structures (Reed, 1999). After this, topology became a particularly important research area in the field of the Geographical Information Science (GISc), for it is a central defining feature of a GIS (Theobald, 2001).

But, what is topology about? In a few words it can be said that in the context of a GIS topology deals with,

> "the subset of spatial relationships characterised by the property of being invariant under topological transformations, such as translation, rotation and scaling." (Clementini *et al.*, 1993).

Laurini and Thompson (1992) stated that topology refers to,

> "relationships between places and spaces from the point of relative positions."

Montgomery and Schuch (1993) defined it as:

> "the aspect in a GIS which allows specialized analytical operations of spatial search and overlay. It refers to the spatial relationships between the points and the lines that define the geographical entities."

DeMers (1997; cited in Theobald, 2001) said that:

> "topology is typically defined as spatial relationships between adjacent or neighbouring features."

To give a classic example, if we consider a rubber sheet with the word "STOP" printed on it, as we stretch it we will never obtain the word "POST". This example shows in particular how the spatial relationships of *adjacency* between the objects *S*, *T*, *O* and *P* are preserved when the rubber sheet is submitted to deformation.

The topological transformation mentioned above may change geometric properties of the spatial objects and even their absolute positioning, but never change the spatial arrangement between them. As far as maps are concerned, and according to Laurini and Thompson (1992), there are four properties said to be invariant under deformation in the base:

- Incidences (two nodes per line, or lines at nodes),
- Intersections,
- Adjacencies (neighbours for areal units),
- Inclusions (points in polygons).

Topology has long been a key GIS requirement for data management and integrity (ESRI, 2005). Functionality to implement topological analysis of 2D data has been provided by many commercial vendors, including Environmental Systems Research Institute (ESRI, 2004) and Intergraph (Intergraph, 2004) amongst others. This functionality is well established, and provides a set of rules and behaviours that model how points, lines and polygons share geometry (ESRI, 2005). Topology constitutes an important component of the analytical capabilities of a GIS, such as overlay analysis, networks analysis and data quality control, and is utilised in a wide variety of fields ranging from urban/regional planning to geology, or archaeology to

transportation (Laurini and Thompson 1992; Reed, 1999; Theobald, 2001; ESRI, 2005).

According to ESRI (2005), topology is employed in a GIS to:

- Manage shared geometry (*i.e.* constrain how features share geometry; for instance, how land parcels share boundaries, how street centrelines and the boundaries of census blocks share geometry, etc.);
- Define and enforce data integrity rules (*i.e.* no gaps should exist between parcel features, parcels should not overlap, road centrelines should connect at endpoints);
- Support topological relationship queries and navigation (*i.e.* have the ability to identify adjacent and connected features, find the shared edges, and navigate along a series of connected edges);
- Support sophisticated editing tools that enforce the topological constraints of the data model (*i.e.* ability to edit a shared edge and update all the features that share the common edge);
- Construct features from unstructured geometry (*i.e.* the ability to construct polygons from lines sometimes referred to as "spaghetti").

2.2 Mathematical topology

In mathematical terms, topology has been called "the study of continuity" (Frank and Kuhn, 1986). It focuses on the properties that are preserved under certain transformations of the space, called *homeomorphisms* (Frank and Kuhn, 1986; Worboys, 1995).

A *homeomorphism*, or *topological transformation*, of $\mathbb{R}^2$ is a *bijection* of the plane (*i.e.* a function that is an *injection* – an *one-to-one* relationship, mapping distinct objects in the domain to distinct objects in the range; and a *surjection* – an *onto* relationship that makes each distinct object in the range the image of at least one object in the domain). This bijection transforms each neighbourhood in the domain into a neighbourhood in the range (injection); furthermore, any neighbourhood in the range must be the result of the application of the transformation to a neighbourhood in the domain (surjection) (Frank and Kuhn, 1986; Worboys 1995, pg 120).

The mathematical study of the topological transformations can be split into two main branches: pointset (or analytic) topology; and algebraic topology (Frank and Kuhn,

1986; Worboys, 1995). The former branch, as its name tells us, deals principally with point sets and the related concepts of neighbourhood, nearness and open sets; the latter focuses on the theory of simplicial complexes and it is also known as combinatorial or geometric topology.

2.2.1 Pointset topology

A topological space can be defined in several different ways. Worboys (1995) presents a particular definition based upon the primary notion of *neighbourhood*. In a few words, given any set of points a collection of its subsets - constituting the neighbourhoods - is defined, providing the so called *neighbourhood topology* on the set. More formally, let *S* be a set of points. A *neighbourhood topology* is a collection of subsets of *S*, called *neighbourhoods*, which satisfy the two following conditions:

(*C1*) Every point *p* in *S* is in some neighbourhood (*i.e.* any point belongs at least to one subset of *S*, apart from being a member of *S* itself);

(*C2*) The intersection of any two neighbourhoods of any point *p* in *S* contains a neighbourhood of *p* (*i.e.* when two neighbourhoods of a point *p* intersect, the resulting intersection contains another neighbourhood of the same point *p*).

Figure 2.1 shows conditions *C1* and *C2* of a topological space as described above. As illustrated in the diagram, different neighbourhoods are shown surrounding each point in the set, and two overlapping neighbourhoods are shown containing another neighbourhood in their intersection.

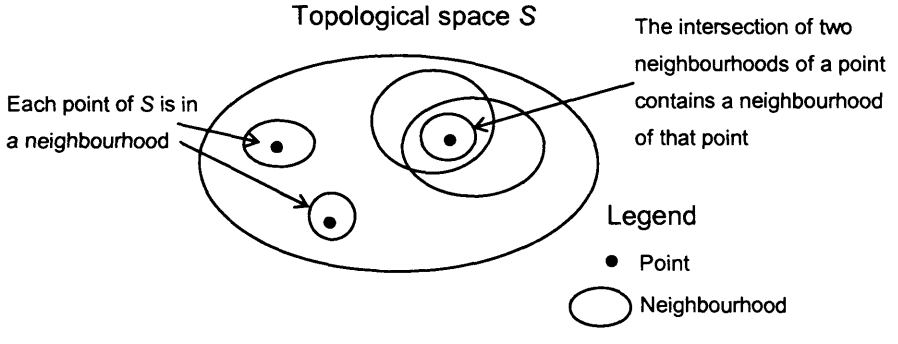


Figure 2.1 – Points and their neighbourhoods in a topological space. (After Worboys, 1995)

In topological terms, a set of points is open if it does not contain its boundary, otherwise it is said to be closed. Based on this concept, the 2-dimensional embedding

space $\mathbb{R}^2$ is a topological space if the neighbourhood of a given point p in $\mathbb{R}^2$ is defined to be any open disc that has p within it. The concepts of *interior*, *boundary* and *exterior* of a pointset have been applied to develop different topological frameworks. Frameworks allow the identification of all possible topological relationships between objects (also, Worboys, 1995; Ellul, 2004), as described above.

The most common and important example of a topological space, for the purposes of a GIS, is the so called *usual topology* for the Euclidean plane; so called because this is the topology which naturally comes to mind with the Euclidean space, and corresponds precisely to the rubber-sheet topology mentioned earlier. The rubber-sheet transformations, which stretch and distort the plane but never fold it up nor tear it apart, are called, and defined in a more formal way, as homeomorphisms. Given our purposes, this concept is applied here only with reference to $\mathbb{R}^2$. However, this is a much more general concept, which can be applied to any topological space (Worboys, 1995).

Many ideas implicit in mapping are based upon this concept of homeomorphism (Worboys, 1995). To illustrate the concept of homeomorphism, let us consider the London Underground Map which is a familiar map, and hence appears to be fairly straightforward. Figure 2.2 shows part of the London Underground network (corresponding to central London) embedded in a geographical map. To obtain a more concise and simpler map, much better known amongst commuters and tourists (shown in Figure 2.3), a topological transformation is applied to the original map. The two maps are different in geometry, but they can be said to be topologically equivalent. In fact, the original map was distorted (stretched or shrunk), *i.e.* the absolute position of the stations changed as well as the length of the underground lines, but it should be noticed that the position of the stations relative to each other as well as the valence of junctions, remained intact. In this context, these two maps are said to be homeomorphic.

Figure 2.2 - The London Underground map embedded in a geographical map. (Copyright fourthway.co.uk; From TfL, 2005)

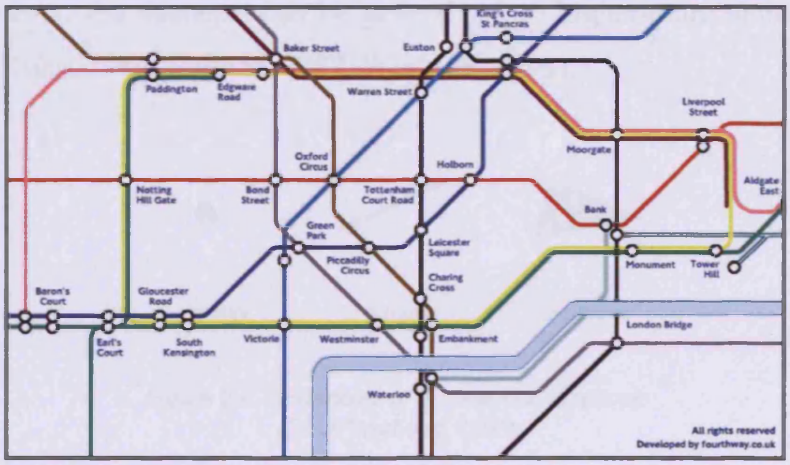


Figure 2.3 – The London Underground map - a topological transformation of the original diagram embedded in a geographical map. (Copyright fourthway.co.uk; TfL, 2005)

2.2.2 Algebraic topology

A typical result of this branch of mathematical topology, also known as combinatorial or geometric topology, is Euler's very well known formula (Gibbons, 1989; Laurini and Thompson, 1992; Worboys, 1995; Wilson, 1996):

> given a polyhedron with a total number of faces, edges and vertices, f, e and v respectively, then
>
> $$f - e + v = 2.$$ *(Equation 2.1)*

A very similar formula can be derived and applied to an arrangement of cells in the plane, with arcs forming the boundaries and nodes forming the intersection points of

the arcs (Gibbons, 1989; Laurini and Thompson, 1992; Worboys, 1995; Wilson, 1996):

> given a cellular arrangement in the plane with a total number of faces, edges and vertices, f, e and v respectively, then
>
> $f - e + v = 1$. *(Equation 2.2)*

Algebraic topology is fundamentally based on the theory of *simplicial complexes* (Worboys, 1995). The formal model which uses this theory is the key for the implementation of topological systems within the context of GIS. In the 2-dimensional case, simplicial complexes are simple triangular network structures in the Euclidean plane. It should be noticed that the constructions in this section are all planar; however, the concepts can be generalised to higher-dimensional structures (Frank and Kuhn, 1986; Carlson, 1987; Worboys, 1995).

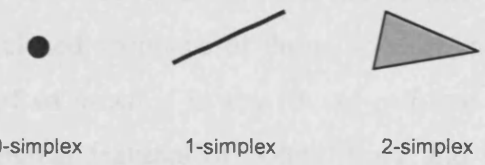


Figure 2.4 – Examples of 0-, 1- and 2-simplices. (From Worboys, 1995)

- A *0-simplex* is a set consisting of a single point;
- A *1-simplex* is a closed finite straight-line segment (also defined as the convex hull of two nodes in general position);
- A *2-simplex* is a set consisting of all the points on the boundary and in the interior of a triangle whose vertices are not collinear or coincident (also defined as the convex hull of three nodes in general position);
- More generally, a face of an *n-simplex* is the convex hull of a nonempty subset of n nodes in general position (in the n-dimensional space).

Simplices are the building blocks of larger and more complex structures, the simplicial complexes. A simplicial complex C is a finite set of simplices satisfying the following properties (Frank and Kuhn, 1986; Worboys 1995):

- A face of a simplex in C is also in C;
- The intersection of two simplices in C is either empty or is also in C.

This means that: if a simplex is an element of a simplicial complex then all faces of the simplex are elements of the simplicial complex as well; second, the simplices in a simplicial complex do not overlap.

In more general terms, a simplicial complex can be made up of a combination of any simplices, and thus can be used to represent real world geometrical objects. However, this approach appears to be weak when representing real phenomena. Indeed, simplices are defined as convex hulls and real phenomena may require the representation of non-convex areas and non-straight lines as well. This can be achieved by a generalisation of the simplex concept (Frank and Kuhn, 1986).

As explained in section 2.3, Egenhofer and Herring (1990) proposed a slightly different approach based on the concept of cells. The use of cells addresses the shortcomings of simplices in that regard. A *0-cell* is a node (the minimal 0-dimensional object); a *1-cell* is the link between two distinct *0-cells*; and a *2-cell* is the area described by a closed sequence of three, or more, non-intersecting *1-cells*. More generally, a *face* of an *n-cell A* is any *(0...n)-cell* that is contained in *A*. The commonly used geographical features of points, lines and regions are defined as follows (Egenhofer and Herring, 1990; Theobald, 2001):

- A *point* is a single *0-cell* in $\mathbb{R}^2$;
- A *line* is a sequence of connected *1-complexes* in $\mathbb{R}^2$ such that they neither cross each other nor form closed loops,
    - A *simple line* is a line with two disconnected ends (Figure 2.5*a*),
    - A *complex line* is a line with more than two disconnected ends (Figure 2.5*b*);
- A *region* is a *2-complex* in $\mathbb{R}^2$ with a non-empty connected interior,
    - A *region without holes* is a region with a connected exterior and a connected boundary – also called *region with connected boundaries* (Figure 2.5*c*),
    - A *region with holes* is a region with a disconnected exterior and a disconnected boundary (Figure 2.5*d*).

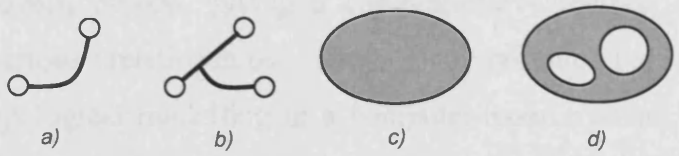


Figure 2.5 - a) a simple line and b) a complex line;
A region with c) connected boundary and d) disconnected boundary.
(From Egenhofer and Herring, 1990)

This cell-based data model differs from the simplicial data model in one particular property: simplices are convex hulls whereas cells may have an arbitrarily shaped interior. The relevant topological primitives of cell complexes are based upon four properties: closure, boundary, interior and exterior (Egenhofer and Herring, 1990).

Since the early 1970s, different topological models have been developed in GIS upon this concept of "nodes, arcs and cells" to model real world data. According to Maguire and Dangermond (1991), and Raper and Maguire (1992) (cited in Theobald, 2001), two main types of topological data structures (TDS) can be identified: *directional* and *complex*. An example of a directional approach is the Dual Independent Map Encoding (DIME) system, developed to process the 1970 USA Census. DIME is widely considered to be the first explicit TDS, and relied on the application of topology to reduce the "optical clutter" of common boundaries represented by duplicate lines and to detect data entry errors in the database (Cooke and Maxfield 1967, Peucker and Chrisman 1975, Clarke 1990, Chrisman 1997, all cited in Theobald 2001; Laurini and Thompson, 1992). Considered to be a further innovation in TDS, POLYVRT is an example of a complex TDS. It was designed to handle more complicated features by replacing a single-line segment (which DIME used) with a chain to represent an arc. It also explicitly stored adjacent polygons (Peucker and Chrisman, 1975, cited in Theobald 2001).

2.3 Topological frameworks

A formal understanding of geometrical relationships between spatial objects is a fundamental concept for the analysis of spatial data (Egenhofer and Herring, 1990).

Topological frameworks provide a theoretical means to formally describe spatial relationships between objects, giving a comprehensive method of differentiating between the various relationships. They also provide the groundwork for implementing topological modelling in a computer-based system, as in this case a series of well-understood rules is required to identify and define relationships between objects.

In the literature, several conceptual structures to support topological relationships between spatial objects can be found, proposed by different authors from the early 20th century to the present. Some of these approaches are briefly reviewed in this section.

One of the very first proposals was set up in 1929 by Tarski (cited in Sullivan 1973, and in Sowa 2000), who showed that the concepts of point, line and plane are physically unrealistic as well as theoretically unnecessary to define three-dimensional relationships. The author identified five relationships whose definition is based upon mereology[1], with Sphere as the only geometrical primitive. According to his proposal, given the spheres *A* and *B* they can relate to one another as follows (*A* and *B* are interchangeable):

- *A* is disjoint from *B*;
- *A* can be externally tangent to *B*;
- *A* can be internally tangent to *B*;
- *A* and *B* can be externally diametrical to another sphere *C*;
- *A* and *B* can be internally diametrical to *C*;
- *A* can be concentric with *B*.

Egenhofer and Herring proposed what is known as the 9-Intersection Model for Topological Relations (Egenhofer and Herring, 1990; Egenhofer *et al.*, 1994). It describes binary topological relations between two regions, *A* and *B*, based upon the comparison of *A*'s *interior* (A°), *boundary* (δA) and *exterior* (A^-), with *B*'s *interior* (B°), *boundary* (δB) and *exterior* (B^-). These six object-parts can be intersected such that they form the description of the topological relations between two regions. The possible intersections are:

[1] The theory of parthood relationships, whose principles govern the relevant relations of part-to-whole as well as the relations of part-to-part within a whole (Varzi, 1996, 2003).

- A's interior with B's interior ($A^0 \cap B^0$);
- A's interior with B's boundary ($A^0 \cap \partial B$);
- A's interior with B's exterior ($A^0 \cap B^-$);
- A's boundary with B's boundary ($\partial A \cap \partial B$);
- A's boundary with B's interior ($\partial A \cap B^0$);
- A's boundary with B's exterior ($\partial A \cap B^-$);
- A's exterior with B's exterior ($A^- \cap B^-$);
- A's exterior with B's boundary ($A^- \cap \partial B$);
- A's exterior with B's interior ($A^- \cap B^0$).

As far as this topological framework is concerned, an emphasis is given in this thesis on the topological relations between regions without holes and with connected boundaries. This is mainly because this research focuses in particular on the topological relations between polygons. According to Egenhofer and Herring (1990), eighteen relations exist in $\mathbb{R}^2$ if the region boundaries are connected or disconnected, eight of which can be realised only for regions with connected boundaries. The authors verified the existence of the topological relations corresponding to the 9-intersections by finding their geometric interpretation. Figure 2.6 depicts prototypes of the eight relations between arbitrary regions. They have been called: *disjoint*, *contains*, *inside*, *equal*, *meet*, *covers*, *coveredBy* and *overlap* (Egenhofer and Herring, 1990; Egenhofer *et al.*, 1994).

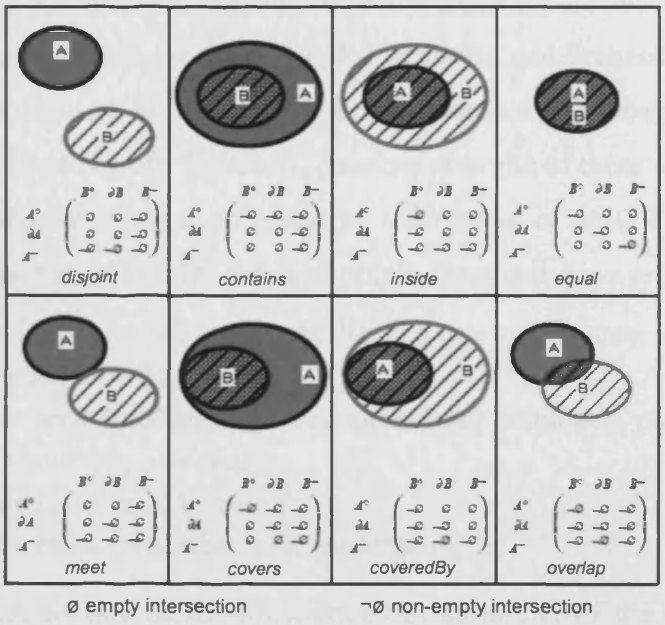


Figure 2.6 – The 9-Intersection Model: a geometric interpretation of the eight topological relations between two spatial regions with connected boundaries and without holes. (From Egenhofer and Herring, 1990)

A different approach was proposed by Cohn *et al.* (1997). These authors defined eight different relations whose set constitutes the so called RCC8 (*Region Connection Calculus*). Given two regions, x and y, the following eight relations are possible ("i" stands for "inverse situation"):

- DC(x, y) - x is disconnected from y;
- EC(x, y) - x is externally connected to y;
- PO(x, y) - x partially overlaps y;
- TPP(x, y) - x is a tangential proper part of y;
- TPPi(x, y) - y is a tangential proper part of x;
- NTPP(x, y) - x is a nontangential proper part of y;
- NTPPi(x, y) - y is a nontangential proper part of x;
- EQ(x, y) - x is identical with y.

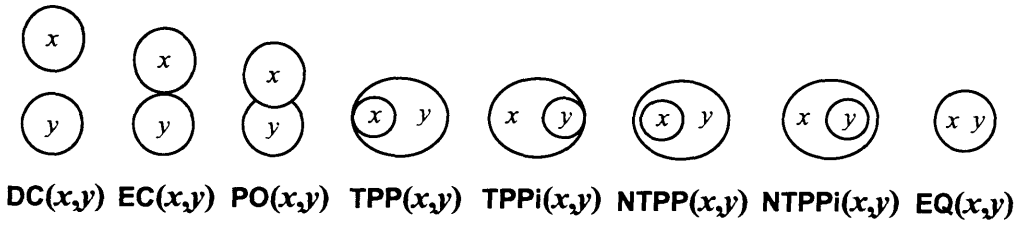


Figure 2.7 – Illustrations of the eight RCC8 relations. (From Cohn et al. 1997)

According to Cohn *et al.* (1997), several authors proposed similar approaches to the RCC8 that should be applied in the context of GIS (Egenhofer and Franzosa 1991; Egenhofer 1991; Clementini, Di Felice and Oosterom 1994; Egenhofer 1994; Egenhofer, Clementini and Di Felice 1994; Egenhofer and Franzosa 1995; Haarslev and Möller 1997), though all of them are firmly based on a point-set approach rather than RCC8's logic-of-regions approach. Another example of these approaches, the so called *Calculus-based Method*, proposed by Clementini *et al.* (1994), is mentioned below. Five distinct relationships between points, lines and areas are identified:

- Touch (between two areas; two lines; a line and an area; a point and a line; a point and an area);
- In (area/area; line/line; line/area; point/line; point/area, point/point);
- Cross (line/line; line/area);
- Overlap (area/area; line/line);
- Disjoint (area/area; line/ area; point/point).

Given the focuses of this thesis, illustrations in particular of the *touch* and the *in* relationships between two areas are included in Figure 2.8 and Figure 2.9, respectively.

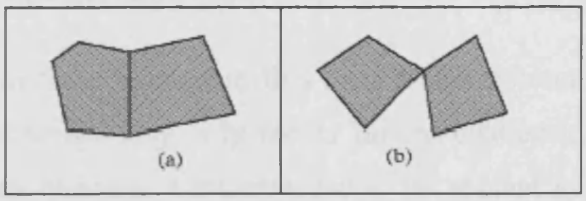


Figure 2.8 – The Calculus-based Method: topological situations illustrating the "touch" relationship between two areas. (From Clementini et al., 1994)

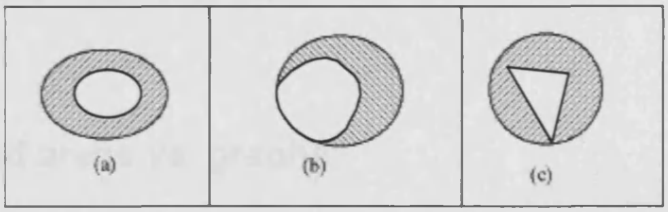


Figure 2.9 - The Calculus-based Method: topological situations illustrating the "in" relationship between two areas. (From Clementini et al., 1994)

Other authors, including Laurini and Thompson (1992), and McDonnell and Kemp (1995) (cited in Theobald, 2001), identified only three topological properties between spatial objects, namely: adjacency; containment; connectivity.

As has been noted, and is explained in detail in Chapter 6, this research focuses on the topological relations between polygons. In particular, the topological relationships of *adjacency*, *touching* and *containment* between polygons are of main interest (*vd.* Chapter 6).

Because the Egenhofer and Herring 9-Intersection Model is the most commonly recognised topological framework in GIS, the concepts of its topological relationships of *meet* and *contains* were borrowed for our purposes. However, in the context of this research, the *meet* relationship needs to be subdivided in two different topological situations: when two areas share at least one arc, the relationship is called in our case *adjacency*; if the two areas happen to meet at a node, the relationship is distinguished from the previous one by being called *touching*.

Alternatively, the concepts of the *touch* and *in* relationships can also be borrowed from the Calculus-based Method proposed by Clementini *et al.* (1994). Likewise, the

topological situations (a) and (b) of the *touch* relationship (illustrated in Figure 2.8) need to be discriminated and described as: (a) *adjacency* and (b) *touching*.

Some of the frameworks presented in this section are commonly associated with topology in GIS. Although they only model binary relationships between spatial objects, some of their concepts were extended to be applied in the context of this research. This was principally because one of the focuses of this research is the investigation of the topological relationships between multiple objects across networks of repeated binary relationships.

2.4 Networks and areas *vs.* graphs

A graph is said to be the most basic simplicial complex (Frank and Kuhn, 1986; Worboys 1995). Simplicial complexes are therefore a generalisation of graphs; they include graphs as a special case in which the complex is made constructed from 0- and 1-simplices, *i.e.* vertices and edges. Because cell complexes are in turn said to be a generalisation of simplicial complex (as noted in section 2.2.2), thus cell complexes can be considered a generalisation of graphs as well.

In the real world some geographical entities can be represented for topological analysis purposes by *vertices*, and their connections (spatial, or topological, relationships) by *edges*. The combination of vertices and edges forms a network that can be viewed as a *graph*. In such a topological data structure, information regarding, for instance, line shape, compass orientation or line length, is normally thrown away, concentrating on the structural components: junctions and connections (Laurini and Thompson, 1992).

2.4.1 Topological consistency

Let us consider a general map with a set of connected polygonal regions representing, say, various administrative territorial units. This kind of a map comprises vertices, edges and spaces; the diagram in Figure 2.10 (on the left) is a prototype of such a map. Because of the "node, edge, face" model derived from graph theory, this map can also be treated as a planar topological map and

"translated" by a planar graph (*vd.* Chapter 3, section 3.3), provided the fact that there are no overlapping regions nor holes and no line crossings without producing vertices (*vd.* Figure 2.10) (Laurini and Thompson, 1992; ESRI 2005).

For the topological representation of a planar set of polygons, let us refer to a very often used terminology adopted by the USA Census Bureau (also cited in Laurini and Thompson, 1992) which comprises five rules for vertices, edges and areal cells (it should be noted that the definition of *cell* is identical to that given in Egenhofer and Herring, 1990):

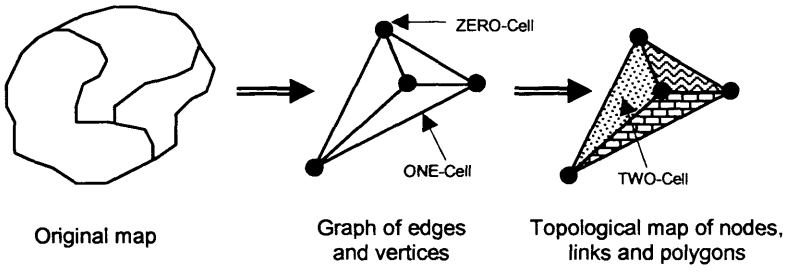


Figure 2.10 – Maps and respective graphs; the USA Census Bureau topological cells. (From Laurini and Thompson, 1992)

(i) Every 1-Cell has two 0-Cells;

(ii) Every 1-Cell has two 2-Cells;

(iii) Every 2-Cell is surrounded by a chain of alternating 1-Cells and 0-Cells;

(iv) Every 0-Cell is surrounded by a chain of alternating 1-Cells and 2-Cells;

(v) There are no intersections that are not 0-Cells.

The 0-, 1- or 2- cells correspond to the geometric entities of points, lines and areas, respectively. Isolated points will still be points, whereas points that are topological junctions are 0-cells (nodes). Disconnected or unrelated lines features are not 1-cells; this term refers to the bounding edges of polygons.

Conditions *(i)* and *(ii)* state that every line that is a 1-cell has two ends and two associated areal spatial units. They also represent a special condition of the duality of 0- and 2-cells for the edges. Conditions *(iii)* and *(iv)* refer to the completeness of the mixture of entities: a cycle around every vertex should encounter a succession of edges and spaces; a cycle around a polygon should consist of a series of alternating edges and nodes. The fifth condition, *(v)*, means that the geometry of the set of

polygons does not contradict topology, *i.e.* if two 1-cells cross then there should be a 0-cell defined (Laurini and Thompson, 1992).

It is important to recall at this stage that not all of a cartographic map's features, like the curvature of lines, the shape of polygons or the labels for places, are preserved under deformation. Indeed, lengths and angles can change, shape of features can vary and also the absolute placement of objects may differ, but not their neighbourhood relationship. In other words, the four properties of *incidences*, *intersections*, *adjacencies* and *inclusions*, are invariant under deformation in the base (Laurini and Thompson, 1992; *vd.* section 2.1).

Topology deals actually with these four properties of the maps that are invariant under deformation of a surface. But topology may not necessarily be logically internally consistent (*e.g.*, an edge may be missing between two vertices). According to Laurini and Thompson (1992), a *topologically consistent map* (or *database*) is defined as "having all spatial entities projected upon a plane surface, with no freestanding features, and having complete topology", which means the five rules identified above, *(i)* to *(v)*, are obeyed. *Completeness of inclusion* means that there are no isolated non-connected points and that all lines are parts of boundaries of polygons (note that these polygons do not necessarily mean complete geographical features); the *completeness of incidence* means that lines intersect only at points associated with the ends of lines. All these consistency conditions may be seen also as constraints for the integrity of a cartographic information database.

For the automation of the rules explained above, graph theory appears to be useful. Indeed, besides counting of the number of vertices, edges and sub-graphs, measures for properties of connectivity are also available to answer questions like: given the physical distance or even the travelling time between two vertices in a graph, how far are they from each other? What is the relative level of accessibility of different cities served by a network of air routes, railways or roads? (Laurini and Thompson, 1992)

Graphs may be compared by their count of vertices, edges and circuits. Also other useful descriptive tools can be provided by devising derived and normed measures (Laurini and Thompson, 1992). For instance, particular vertices may be compared by counting the number of their incident edges, being referred to as the *degree* or

valence of the vertex. Furthermore, the conditions of connectivity for the whole network may be summarised by a count or average vertex degree, or may use measures based on edges, like the minimum or maximum number of edges for a given number of vertices; the number of edges relative to the maximum possible number; the number of circuits relative to the maximum possible number; or the ratio of edges to vertices (Laurini and Thompson, 1992).

Apart from the descriptive measures for networks, graphs have some special properties representable by a single equation, as Euler proved a few centuries ago (*vd.* Equations 2.1 and 2.2).

2.4.2 Planar enforcement

As explained in previous section, rules *(i)* to *(v)* apply only to planar graphs. Thus, a requirement for planarity is imposed onto 2-dimensional topological data structures. To meet this condition vendors make use of the process of *planar enforcement* which ensures that topological primitives are both contiguous throughout the topological space (*i.e.* there are no holes in between them) and cannot overlap (Laurini and Thompson, 1992; Worboys 1995).

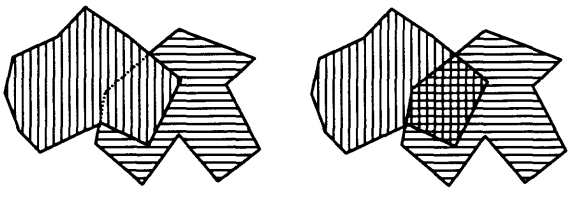


Figure 2.11 – The planar enforcement process.

In Figure 2.11 the process of data planar enforcement consisted of intersecting the two initial polygons. If two polygons overlap this means that they happen to partially occupy exactly the same space at the same time. The operation of intersecting them consists of determining the area which is simultaneously occupied by both polygons, and splitting them into three different polygons.

The process of breaking down overlapping objects into their constituent parts, and then determining whether any of them share common components of topology (*i.e.*

intersect), brings additional complexity onto the cell-based topological data structure, as illustrated in Figure 2.12.

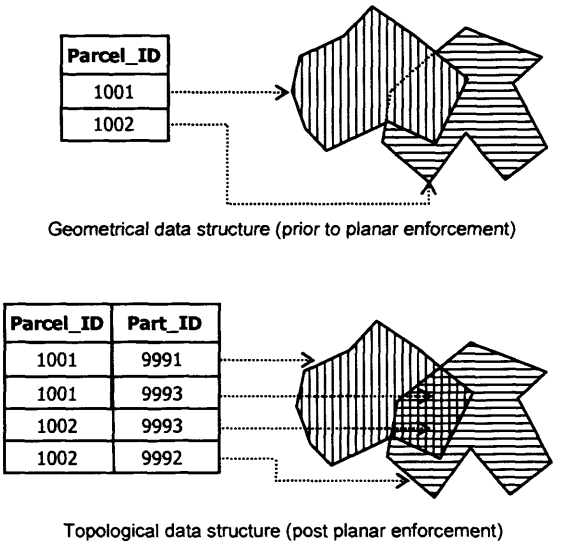


Figure 2.12 – Additional complexity added by planar enforcement. (After Rigaux et al., *2004 and ESRI, 2005)*

Though "intersection" is a valid topological relationship (Egenhofer and Herring, 1990), the issue is that it cannot be directly supported by the simplex nor cell-based topological data structures described in section 2.2.2, given that the planar enforcement does not model overlapping objects.

The complex features *route*, *section* and *region*, supported by the ESRI coverage topological data structure, are an attempt to model the "intersection" relationship between objects (Rigaux *et al.*, 2002).

Routes define paths along an existing set of arcs. They can be disconnected. Figure 2.13 depicts two different examples of routes: a route can start and end at an arc end node *(a)*; or may start/end at a vertex along an arc *(b)*. Routes are made up of *sections*.

A *section* is an arc or a portion of an arc. A section cannot overlap two arcs. It is described in the data model by the respective arc identifier, the percentage of the arc used, and by the distance along the route the section represents. Routes and sections can be used, for instance, for mapping bus routes on a road network.

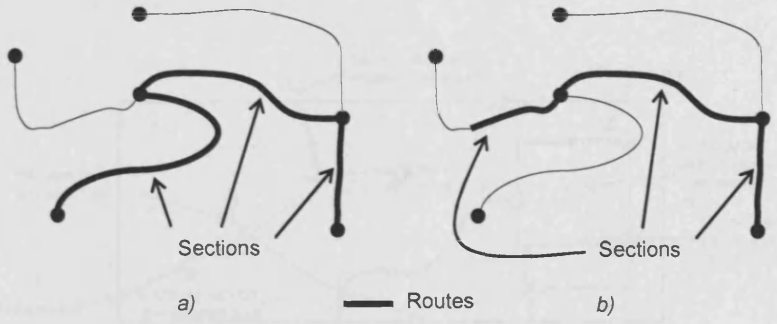


Figure 2.13 – Examples of routes and sections in the ESRI's coverage data model: a) route starting and ending at an arc end; b) route starting at a point along an arc. (After Rigaux et al.*, 2002)*

A *region* is a set of polygons that can be connected or not. Several regions may share the same faces. In that case, two regions of the same coverage may overlap. In Figure 2.14, five polygons and three regions are depicted; polygon *2* is shared by region *A* (which also includes polygon *1*) and region *B* (which also includes polygon *3*); in turn, region *C* consists of two disconnected polygons, *4* and *5*. Regions can be used for representing composite geographic objects, such as a country comprising the main land and a couple of islands.

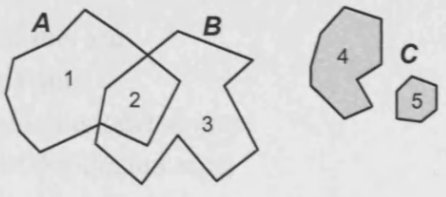


Figure 2.14 – Examples of regions in the ESRI's coverage data model. (After Rigaux et al., 2002)

2.5 Digital cartography

The main avenue contributing to the current use of topology in geographic information systems is the one of *cartography*, comprising digital cartography and data capture processes in general. More specifically, the digitizing process of transforming paper maps into digital format involves tracing the edges of all its geographical entities (buildings, land parcels, road networks, etc.). Typical errors of the digitizing procedure, especially in the manual process, are duplicated or omitted nodes, the so called *overshoot* and *undershoot* of lines beyond other lines or a map edge, duplicated or omitted lines, sliver polygons, amongst others.

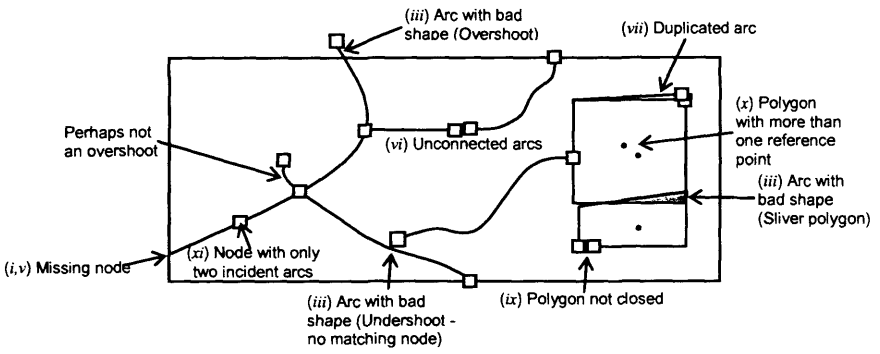


Figure 2.15 – Some conditions for digitized maps. (After Laurini and Thompson, 1992)

The diagram in Figure 2.15 contains examples of possible conditions that may result from preparing digitized versions of maps. These conditions can be divided into geometric and topological conditions (Laurini and Thompson, 1992; ESRI 1995):

Some geometric conditions are:

(*i*) A node is missing or is misplaced;
(*ii*) An arc is missing or is misplaced;
(*iii*) An arc has a bad shape or too many (or too few) points on it; or coordinates are missing or incorrect;
(*iv*) A node has more than one position.

Some topological conditions are:

(*v*) A node is missing;
(*vi*) Existence of unconnected arcs;
(*vii*) Existence of duplicated arcs;
(*viii*) A polygon may be missing;
(*ix*) A polygon has a gap between two arcs, *i.e.* it is not closed;
(*x*) A polygon has no, or more than one, reference point associated with it;
(*xi*) A node has fewer than three incident arcs (excluding particular cases where this situation should occur).

Such topological conditions may arise as a consequence of imperfect geometry. It may happen sometimes that is not possible to validate whether a certain condition is inherently geometrical or topological, or whether it represents an actual error. Tarle (1995) proposes four levels of topological validity of spatial data:

- Digital Cartography – the end products are familiar maps where, for instance, the path of a river is interrupted by a bridge, and map features are identified by their colour and shading. Features do not interconnect and the data cannot be used for analysis without extensive manipulation.
- Limited Implicit Topology – map features that are supposed to intersect actually do, but no junction node is created and the spatial element that is

being intersected remains intact. Areas are made up from lines of unrelated graphic elements. Duplicate data are not acceptable. Map feature codes are contained in the database record rather than in the graphics/shading.

- Full Implicit Topology – spatial data is structured by following a series of rules, and connectivity implies that junctions or nodes are created where objects intersect. Each feature is broken up into multiple entities as a result of this rule. Areas are implicit as boundaries are formed from closed polygons.
- Full Topology – this is the highest level of spatial data structure with full attribute and topological relationships. Lines intersect and connect at exact nodes. Each edge or line has a unique identifier in the database. Areas are explicit and are constructed through explicit linkages to the lines and nodes bounding them. Duplicate data are not acceptable.

There are several advantages in applying topology to the process of collecting data, not only in terms of improving data quality but also due to reduced storage requirements. The ability to explicitly store relationships between adjacent features is probably one of the main advantages of topological data structures (TDS). Therefore, boundaries shared by two adjacent polygons do not need to be stored twice. As a consequence, adjacency, containment and connectivity analyses performance is considerably improved. In addition, errors introduced during map digitizing and data entry can be automatically checked (DeMers, 1997; Burrough and McDonnell, 1998; cited in Theobald, 2001).

But TDS also have some disadvantages. Aronoff (1989; cited in Theobald, 2001) pointed out that "topological tables have to be created beforehand, requiring computational time and also multiple editing sessions to remove under-, overshoots and sliver polygons". Computational time is not so much of a problem any longer as computers are generally much faster; however, because the chains of vertices that constitute geographical features are not stored sequentially, the graphic display of them may be slower since they have to be extracted from different data structures or files (Bonham-Carter, 1996; cited in Theobald, 2001). Moreover, 3-dimensional geographical features, such as overpasses and tunnels, and complex features, like one-way streets, self-intersecting transportations routes (*e.g.* bus routes) as well as parcels made up of disjoint polygons, cannot be represented by planar graphs (Raper and Maguire, 1992; cited in Theobald, 2001). These issues led to composite-feature models, like ESRI's *route*, *section* and *region* in coverages (*vd.* section 2.4.2).

As opposed to the TDS, there are the cartographic data structures (CDS). Three major disadvantages are typically cited: first, adjacent polygons that share common boundaries duplicate common vertices, and therefore the file sizes are expected to be larger (Burrough and McDonnell, 1998; cited in Theobald, 2001); second, adjacency, containment and connectivity analyses are severely limited (Cowen, 1988; Maguire and Dangermond, 1991; cited in Theobald, 2001); third, robust map creation and editing cannot be accomplished because of the spaghetti digitizing as a result of standard digitizing procedures (Theobald, 2001).

In spite of this, an advantage of CDS over TDS is that they can be drawn and edited faster because all the geometric features are stored sequentially in only one file (Theobald, 2001). Most importantly, is that CDS are capable of handling complex polygons, like polygons with islands, because "rings" defining the polygon are stored with their vertices in clockwise order (anticlockwise for the hole arcs), so that the area to the right (as one "walks" along the boundary) is inside the polygon and the left is outside (*e.g.* the ESRI's *shapefile* data model).

2.6 The ESRI's coverage data model

Given the GIS software package available, ESRI's ArcGIS, it is pertinent to review in more detail the topological data structure implemented by this software: the ArcInfo coverage data model, also known as the georelational data model. It was implemented in the first ESRI commercial GIS software, ArcInfo, which was introduced in 1981.

Typically, an ArcInfo database contains several coverages, one per theme each representing a single or a set of geographic objects, such as roads, land-use parcels, post boxes, etc. More precisely, a coverage can be one of the following (Rigaux *et al.*, 2002):

- A set of points;
- A non-planar graph, which means a set of possible intersecting lines (called *arcs* in the GIS terminology);
- A planar graph, that is a set of *arcs* and *nodes* (in the GIS terminology) with the following constraints – the nodes must either start or end an arc, and two arcs can only intersect each other at their ending nodes;

- A set of arcs, nodes and polygons. As before, these objects constitute a planar graph in which the graph facets are also of interest – a polygon is a facet of the graph, and arcs have left and right facets.

The relationships between these different geographic objects are described and stored in the data model using topology. In ArcInfo, the major topological concepts are (ESRI, 1995; Rigaux *et al.*, 2002):

- Arcs connect to each other at nodes (connectivity);
- Arcs that connect to surround an area define a polygon (area definition);
- Arcs have direction and left and right sides (contiguity).

Because of the *arc-node topology* (connectivity), arcs are stored in a coverage as a sequence of points (*x,y* pairs), called in the GIS terminology *vertices*, which define the shape of the arc. The endpoints of the arcs are called nodes. Each arc has two nodes: a *from-node* and a *to-node* (represented by their ID in the columns FromNODE# and ToNODE# respectively – *vd.* AAT table in Figure 2.16). Therefore, each arc is implicitly assigned a direction (from-node → to-node) and ArcInfo maintains a list of polygons on the left and right sides of each arc (*left-right topology*, contiguity). Because of the left-right topology it is possible to know, for instance, which polygons are adjacent to which (ESRI, 1995; Rigaux *et al.*, 2002). This aspect is particularly important in this study, as shown in Chapter 6.

In a coverage, the feature boundaries and points are stored in a few main files that are managed and owned by ArcInfo Workstation, a software component of ArcGIS.

The ARC file holds the linear or polygon boundary geometry as topological edges, called arcs. The AAT info file along with the ARC file (both files related to one another through the common field ARC#, where the feature IDs are stored) store all data for arcs. The PAL file along with the ARC file (both files related to one another through the common field ARC#), store data related to polygon boundaries. In turn, The PAL file along with the PAT info file (both files also related to one another through the common field POLY#), store all data for polygons (ESRI, 2005).

Some of these files are also used to define the topological relationships between each of the edges and polygons. For example, the PAL file lists the order and direction of the arcs in each polygon. In ArcInfo, software logic is used to assemble the coordinates for each polygon for display, analysis, and query operations. The ordered

list of edges in the PAL file is used to look up and assemble the edge coordinates held in the ARC file. The polygons are assembled during runtime when needed (ESRI, 2005).

The LAB file holds point locations, which are used as label points for polygons or as point features, such as a set of points representing oil well locations.

When nodes are used to represent point locations on linear features, a node attribute table (NAT) is used.

As far as composite features are concerned, RAT file refers to a route attribute table, SEC to a section attribute table. Regions have their own attribute table distinct from the Polygon Attribute Table: the PAT-Region subclass attribute table. A fully constructed region coverage has also both: a PAL file which stores the region-arc relationship, and an RXP file which stores the region-polygon relationship.

Figure 2.16 illustrates some of the mentioned files for a simulated map of three polygons, and shows how these files relate to one another.

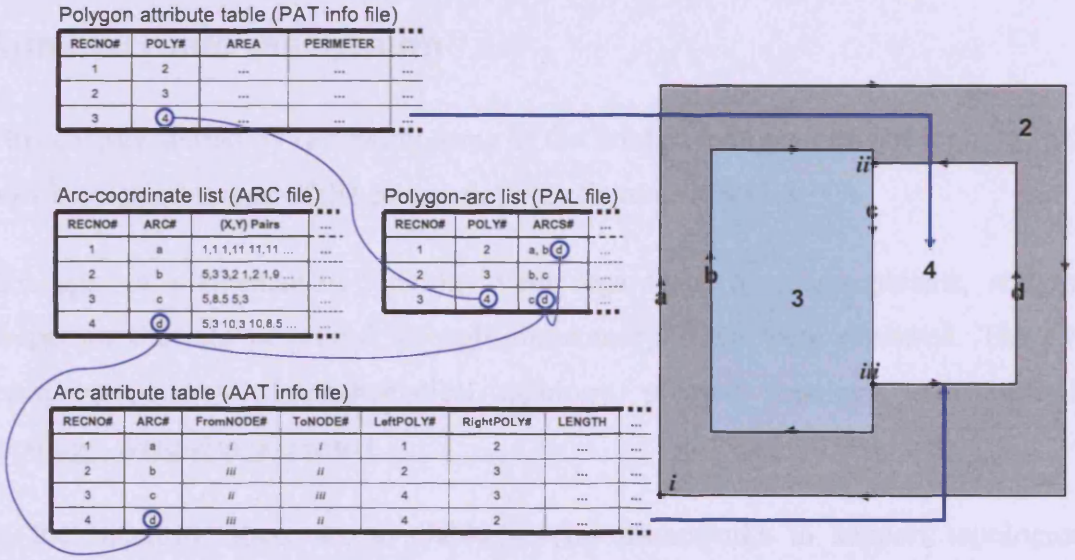


Figure 2.16 – The ArcInfo coverage data model for storing vector data.

The ESRI (2005) identified some advantages of the ArcInfo coverage model:

- It uses a simple structure to maintain topology;
- It enables edges to be digitized and stored once and shared by many features;
- It can represent polygons of various sizes (with thousands of coordinates);

- The topology storage structure of the coverage is intuitive, and its physical topological files are readily understood by ArcInfo users.

But coverages also have disadvantages. These are (ESRI, 2005):

- As noted above, many features have to be assembled on the fly when they need to be used, which makes some operations to be slow. This includes all polygons and composite features, such as routes and regions;
- Topological features (such as polygons, routes and regions) are not ready to be used until the coverage topology is built. If edges are edited topology has to be rebuilt. However, when edits are made to features in a topological dataset, generally a geometric analysis algorithm must be executed to rebuild the topological relationships regardless of the storage model;
- Coverages are limited to single-user editing. Because of the need to ensure that the topological graph is synchronised with the features geometries, only a single user at a time can update a topology. Users would tile their coverages and maintain a tiled database for editing. This enables individual users to "lock down" and edit one tile at a time. For general data use deployment, users would append copies of their tiles to a mosaicked data layer.

2.7 Summary and discussion

This chapter started by reviewing some of the fundamental concepts of topology and how it emerged as one of the central defining features of a GIS.

Concepts of mathematical topology with regards to homeomorphisms, and the properties that are preserved through homeomorphisms, were reviewed. The two main components of mathematical topology, pointset topology and algebraic topology, were also presented.

In the literature, there are available several frameworks to support topological relationships between spatial objects. Some of these frameworks, most commonly associated with topology in GIS, were reviewed in this chapter.

For the purposes of this research, three primitive topological properties of the spatial relationships between polygons are of main interest, namely: *adjacency* (when two areas share at least one arc), *touching* (if two areas happen to meet at nodes), and *containment* (when an area is completely surrounded by another). Two other

topological properties can be intrinsically considered, *connectivity* and *contiguity*, which basically derive from the primitives mentioned above.

Given the GIS software package used in this research, the ESRI's ArcGIS, the ArcInfo coverage data model, also known as the georelational data model, was used for our purposes. Thus, the topological data structure implemented by the coverage data model was described.

For the purposes of preparation of the unstructured data used in this research, contiguity of polygons constitutes a particularly important topological property. TIN triangles with similar characteristics regarding a particular attribute were gathered into polygons whose spatial arrangement was analysed thereafter. This type of spatial discretization entails a complete set of polygonal entities without holes or overlaps, and hence turns our attention to the importance of the concepts and tools that treat the space as a combination of line segments: a network which in turn can be viewed as a graph. The representation of areas and networks as graphs requires both the topological consistency of the data model and the planarity of the objects to be represented. These issues were also discussed in this chapter.

Given the emphasis that has been given to the importance of the role of graph theory in dealing with topological issues, the fundamental concepts of graph theory are reviewed in the next chapter. A few different graph-based applications are covered in order to understand how this mathematical framework can be applied in investigating particular spatial arrangements of entities. In Chapter 3 is also noted how the terminology in graph theory differs from that used in GIS when referring to graphs.

3 Graph theory

As pointed out in the previous chapter, graph theory is a valuable mathematical framework in revealing the spatial structure of geographical entities and their spatial arrangement. Hence, its general concepts are reviewed in this chapter. After briefly introducing this subject in the following section, the basic principles of graph theory and its terminology are summarised in section 3.2. The types of graphs directly used in this research are presented in section 3.3 (*vd.* Appendix L for other types of graphs). Section 3.4 overviews graph-search algorithms used in the process of learning structural properties of graphs. In section 3.5, five examples available from the literature of graph-based applications for spatial system analysis are given: the *eXtended Relational Attributed Graph (XRAG)*, used for the inference of land-use information from a land-cover map; space syntax, which attempts to explain human behaviour and social activities from a spatial configuration point of view; *β-skeletons*, for the description of the internal shape of point patterns; two examples of multiobjective approaches for urban network analysis; and built-form connectivity, which attempts to explain the spatial organisation of an urban system on the basis of its distribution of buildings. This chapter is concluded with a summary in section 3.6.

3.1 Generalities

In our world, many configurations of nodes and their connections representing relationships can occur in a wide range of subjects. They may represent well defined physical networks, *e. g.* man-made like the roads, railways or electrical circuits, or organic molecules, or even less tangible interactions, such as those which might occur in ecosystems, social relationships or databases. Normally, these kinds of

configurations are modelled in formal terms by what are called *graphs*, which are described, from a mathematical perspective, as combinatorial structures. As a matter of fact, formal descriptions of spatial phenomena based on this mathematical framework, are valuable for a clear, concise and consistent definition of the involved spatial relationships (Laurini and Thompson, 1992; Gross and Yellen, 1999).

Graph theory is generally believed to be elegant and is based on only a few basic simple principles, even though they can be disguised in several ways (Temperley, 1981; Bollobás, 1998). Because of their simplicity and due to the usefulness of graphs as models for computation and optimisation, graph theory is a widely applied framework in such diverse research fields as chemistry, biology, geography (and when we talk about geography we are also thinking about areas like environment, ecology, etc.), information technology (such as computer science, GIS, etc.), urban engineering (including for instance, transportation, location/allocation issues, waste management, utilities networks), etc. Graph theory is primarily concerned with maximally efficient flow or connectivity in networks (Bollobás, 1998; Gross and Yellen, 1999).

Besides the theoretical concepts, one of the main purposes of this chapter is also to give an overview of how this mathematical framework has been used so far to address a wide variety of problems, not only in the GIS world (eventually our area of interest) but also in other research fields.

3.2 Basic principles and definitions

In this section, some of the basic principles and terminology that enable us to understand graphs as a framework of combinatorial objects, which represent spatial configurations, are presented. Although some authors argue that the terminology of graph theory is still not standard (like Bollobás, 1998), the one used in this document is well accepted and commonly used throughout the literature reviewed.

To start with, let us consider a general example. Supposing that Figure 3.1 is part of a street map of a certain town, a diagram using points and lines, like the one in Figure

3.2, can represent this situation. Streets are represented by lines e_1, e_2, e_3, e_4, e_5, e_6, e_7 and e_8 and are called *edges*; streets' junctions are represented by points v_1, v_2, v_3, v_4 and v_5 which are called *vertices* and hence correspond to intersections (it should be noted that streets represented by e_7 and e_8 do not cross at the same level, and therefore there is no junction between them); the whole diagram is called a *graph*.

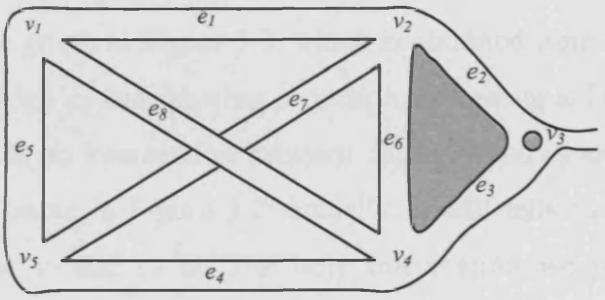


Figure 3.1- An example for part of a street map.
(After Wilson, 1996)

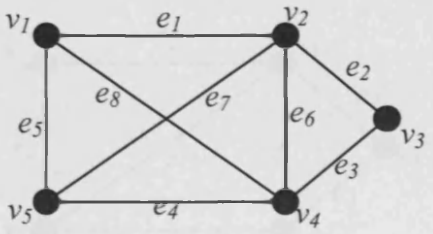


Figure 3.2 - A diagram representing the street map sample above.
(After Wilson, 1996)

At this stage, the terminology presented above for formal mathematical graphs should be distinguished from that commonly used for graphs in the context of a GIS. Thus, for the purpose of this thesis, for graph intersections and their connections within a GIS the terms *node* and *arc* are respectively used.

More formal definitions are presented below.

Graph

A graph G consists of two distinct sets $V(G)$ and $E(G)$ and we write $G=(V,E)$: the elements of $V(G)=\{v_1,v_2,...,v_n\}$ are the vertices (*vs.* nodes) and the elements of $E(G)=\{e_1,e_2,...,e_m\}$ are called edges (*vs.* arcs) representing the interaction between vertices, such that each edge $e_{ij}=v_iv_j$ connects vertices v_i and v_j which are called its

end-points (Temperley, 1981; Gibbons, 1989; Laurini and Thompson, 1992; Wilson, 1996; Bollobás, 1998; Gross and Yellen, 1999).

As noted above, we can see that in the example of Figure 3.1 streets labelled as e_7 and e_8 do not cross each other, supposing, for instance, the existence of a bridge in e_8 over e_7. Therefore, in these particular situations, we can represent the same street network sample by the graph in Figure 3.3, which is obtained from the one in Figure 3.2 by removing the edge e_8 and drawing it outside the rectangle $[v_1v_2v_3v_4]$. We can do that because there is no intersection between edges e_7 and e_8 and hence the new graph obtained is the same in Figure 3.2; actually, it still tells us where the direct road linking junctions v_1 and v_4 is. The only information we have lost regards geometry, namely the location of that road, its length and its straightness. Thus, we can deduce that a graph is a representation of a set of points and how they are joined up, any metrical properties being irrelevant (Wilson, 1996, pg.2). Finally, the graph in Figure 3.2 and Figure 3.3 can also be drawn like the one in Figure 3.4.

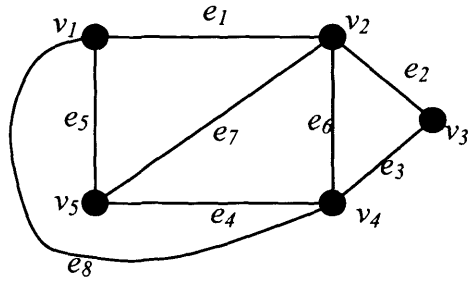


Figure 3.3 - Another possible representation of the street map sample in Figure 3.2. (After Wilson, 1996)

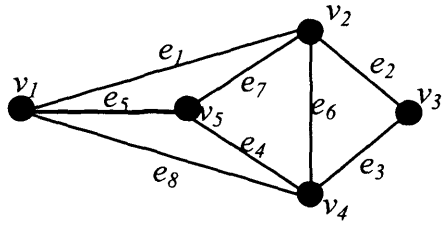


Figure 3.4 - A different aspect of the same graph in Figure 3.2 and Figure 3.3. (After Wilson, 1996)

Finite/Infinite graphs

The number of vertices in a graph shall be denoted by $n = |V|$ and the number of edges by $m = |E|$. If both n and m are finite, as is presumed to be the normal case, then the graph is said to be *finite*; otherwise is called *infinite* (Wilson, 1996, pg. 77).

Self-loop, Multi-edge, Simple graphs

An edge is said to *join* its end-points. A *self-loop* is an edge that joins a single end-point to itself. A *proper edge* is an edge that is not a self-loop. A *multi-edge* (or *parallel edge*) is a collection of two or more edges having identical end-points. Graphs with no self-loop or multi-edges, like the graph in Figure 3.4, are called *simple graphs* (Wilson, 1996, pg.3; Gross and Yellen, 1999, pg.2).

Adjacency

If an edge e has u and v as its end-points, then we say that e is *incident on* v and *on* u. Also, if $(u,v) \in E(G)$, then u is said to be *adjacent on* v and *vice-versa*. For example, in Figure 3.4 e_4, e_5 and e_7 are incident on v_5 which is adjacent to v_1, v_2 and v_4. We also say that two edges are adjacent if they have a common end-point. For instance, taking the previous example, the edges e_4, e_5 and e_7 are adjacent (Wilson, 1996, pg.12).

The *degree* (or *valence*) of a vertex v, $d(v)$, is the number of proper edges incident on v plus twice the number of self-loops (Gross and Yellen, 1999, pg.6). In our example in Figure 3.4, $d(v_3)=2$, $d(v_1)=d(v_5)=3$, $d(v_2)=d(v_4)=4$. The particular case of a vertex v for which $d(v)=0$, means that is an *isolated* vertex.

Another important concept is the *adjacency matrix*, A, of a graph. If the graph has n vertices then A is the $n \times n$ matrix defined such that (Gross and Yellen, 1999, pg.76; O'Sullivan and Turner, 2000):

$$A=[a_{ij}],\ \text{where } a_{ij} = \begin{cases} 1 & \text{if } v_i v_j \in E(G), \\ 0 & \text{otherwise} \end{cases} \qquad \text{(Equation 3.1)}$$

In a simple graph $a_{ij} = a_{ji}$ in all cases. However, as we will see further, in a *directed graph* (or *digraph*) this is not necessarily the case and the symmetry of linkages between vertices may be broken.

In an *undirected graph* with n vertices and m edges, we define its *incidence matrix*, X, as a $n \times m$ matrix such that (Gross and Yellen, 1999, pg.75):

$$X=[x_{ij}],\ \text{where } x_{ij} = \begin{cases} 2 & \text{if edge } j \text{ is a self-loop at vertex } i, \\ 1 & \text{if vertex } i \text{ is incident to edge } j, \\ 0 & \text{otherwise} \end{cases} . \qquad \text{(Equation 3.2)}$$

Neighbours

The concept of *neighbours* of a vertex v_i in V regards the set of vertices $N(v_i)$ which are joined to v_i by edges in E (Gross and Yellen, 1999, pg.50). Formally, we have (O'Sullivan and Turner, 2000):

$$N(v_i) = \{v_j : v_i v_j \in E(G)\} . \qquad \text{(Equation 3.3)}$$

Neighbourhoods

In some cases, it might be useful to think about the *neighbourhood* of a vertex v_i as the sub-graph induced by $N(v_i)$ (Equation 3.3), which can be notated by N_i^* (O'Sullivan and Turner, 2000). This neighbourhood sub-graph consists of the pair of sets given by:

$$N_i^* = < N(v_i) \left\{ v_i v_j : v_j \in N(v_i) \wedge v_i v_j \in E(G) \right\} > \qquad \text{(Equation 3.4)}$$

3.3 Types of graphs

Null graphs

A graph whose vertex and edge sets are empty is called a *null graph* (Wilson, 1996, pg.17; Gross and Yellen, 1999, pg.2).

Sub-graphs, Super-graphs

A sub-graph H of G is a graph obtained from G by removing a number of its edges and/or vertices. The removal of any vertex necessarily implies the removal of every edge incident on it, whereas the removal of an edge does not necessarily imply the removal of its end-points although it may result in one or two isolated vertices. The sub-graph H is defined such that $V(H) \subseteq V(G)$ and the edge set of H, $E(H)$, is any subset of $E(G)$ with the obvious constraint that edges in $E(H)$ may only be a subset of those edges in $E(G)$ which link members of $V(H)$. In these circumstances, it is said that H is *induced* by $V(H)$ and G is called the *super-graph* of H (Gibbons, 1989,

pgs.4-5; Wilson, 1996, pg.12; Gross and Yellen, 1999, pg.48; O'Sullivan and Turner, 2000).

Bipartite graphs

A *bipartite graph* G is a graph whose vertex set V can be partitioned into two disjoint subsets U and W, such that each edge of G has one end-point in U and one end-point in W. The pair (U,W) is called a *vertex bipartition* of G, and U and W are called the *bipartition subsets*. In other words, a bipartite graph is one whose vertices can be coloured black and white in such a way that each edge joins a black vertex (in set U) and a white vertex (in W) (Wilson, 1996, pg.18; Gross and Yellen, 1999, pg.11).

Connected and Unconnected graphs

Some graphs are in two or more parts. A graph is *connected* if and only if a path exists between each pair of vertices and hence if the graph is in only one piece; a graph in more than one piece is an *unconnected* (or *disconnected*) graph which may include though, several connected components or sub-graphs. An important concept is the concept of *disconnecting set*, which is a set of edges in a connected graph G whose removal disconnects G (Temperley, 1981, pg.11; Wilson, 1996, pg.26; Gross and Yellen, 1999, pgs.24, 51).

Walks, Trails, Path graphs and Cycle graphs

Much of graph theory involves *walks* of various kinds. A walk is a way of getting from one vertex to another and consists of a finite sequence of edges, one following another, in which any two consecutive edges are adjacent or identical. The vertex of departure is called the *initial vertex* and the vertex of arrival is called the *final vertex*. The number of edges in a walk is called its *length* (Wilson, 1996, pg.26; Gross and Yellen, 1999, pg.22). For example, in Figure 3.4 $v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_1$ is a walk from v_3 to v_1 of length 3.

A *trail* is a walk with no repeated edges. A trail in which no vertex appears more than once is called a *path*, like the one mentioned in the previous paragraph. But the walk $v_1 \rightarrow v_5 \rightarrow v_4 \rightarrow v_1$ is called a *cycle*. In a more formal way, we say that a *path graph* P is a simple *connected graph* with $|V_P| = |E_P| + 1$ that can be drawn so that all of its vertices and edges lie on a single straight line; an n-vertex path graph is denoted P_n. A *cycle graph* C is a single vertex with a self-loop or a single *connected*

graph C with $|V_C| = |E_C|$ that can be drawn so that all of its vertices and edges lie on a cycle; an n-vertex cycle graph is denoted C_n (Gross and Yellen, 1999, pg.32).

A graph's *diameter*, denoted $diam(G)$, is the longest path between any two vertices in the graph, where the path length between those two points is itself the shortest possible length, and it is given by $diam(G) = max\{d(v,u), \forall v,u \in V(G)\}$. Analogously, the radius of a graph G, denoted $rad(G)$, is given by $diam(G) = min\, max\{d(v,u), \forall v,u \in V(G)\}$ (Gross and Yellen, 1999, pg.38).

Trees and spanning trees

A path with no cycles is a *tree*. A tree that includes all the vertices in the graph is a *spanning tree*. The *minimum spanning tree* is the spanning tree in the graph with the total shortest length (Gibbons, 1989, pgs.40-41; Wilson, 1996, Chap.4; Gross and Yellen, 1999, Chap.4).

Isomorphic graphs

Two graphs G_1 and G_2 are *isomorphic* if there is a *one-to-one* correspondence between the vertices of G_1 and those of G_2 such that the number of edges joining any two vertices of G_1 is equal to the number of edges joining the corresponding vertices of G_2. In other words, it means the conditions of connectedness and adjacency of both graphs correspond, even though their shape may be quite different (Gibbons, 1989, pg.4; Laurini and Thompson, 1992, pg.178; Wilson, 1996, pg.9; Gross and Yellen, 1999, pg.60).

Homeomorphic graphs

Two graphs are defined as *homeomorphic* if both can be obtained from the same parent graph by inserting new vertices of degree 2 into its edges (Gross and Yellen, 1999, pg.261).

Planar *vs.* non-planar graphs; Kuratowski graphs

A *planar graph* is a graph that can be drawn in the plane without crossings, so that no two edges intersect geometrically except at a vertex to which both are incident (Gibbons, 1989, pg.67; Wilson, 1996, Chap.5; Gross and Yellen, 1999, Chap.9). It should seem that the graph in Figure 3.2 is an example of a non-planar graph, for the edges e_7 and e_8 cross each other. But the same graph was redrawn in Figure 3.3 and

Figure 3.4 removing edge e_8 to the outside of the rectangle $[v_1v_2v_3v_4]$ so as to avoid the edges crossing. In this way, we obtained a planar graph in both figures. Moreover, it was proved independently by K. Wagner in 1936 and I. Fáry in 1948 (Wilson, 1996, pg.60) that, with the exceptions of the loops and multi-edges, "every single planar graph can be drawn with straight lines on a 2D plane".

At this stage it must be pointed out that, like the complete graph K_5 and the complete bipartite graph $K_{3,3}$, not all graphs are planar. In addition, note that every sub-graph of a planar graph is planar; and every graph with a non-planar sub-graph is non-planar. The K_5 and the $K_{3,3}$ graphs are typical examples of non-planar graphs since they are taken to be the "building blocks" for non-planar graphs in the sense that every non-planar graph must "contain" at least one of them. Kuratowski's theorem (1930) states precisely that "a graph is planar if and only if it contains no sub-graph homeomorphic to K_5 or $K_{3,3}$" (Gibbons, 1989, pg.77; Wilson, 1996, pg.63; Gross and Yellen, 1999, pg.306). In honour of this mathematician, these two graphs are known as the *Kuratowski graphs*.

Dual graphs

The geometric idea of duality is very old. For example, the "fifteenth book of Euclid", written about AD 500-600, remarks that the dual of a cube is an octahedron, and that the dual of a dodecahedron is an icosahedron (Wilson, 1996, pg. 73).

Given our previous example, the construction of its dual G^*, called the *(geometric) dual of* G, is held in two steps (Wilson, 1996, pg. 73) (*vd.* Figure 3.5):

(i) Inside each face of G, a point v^* is chosen; these points are the vertices of G^*;
(ii) Corresponding to each edge e of G, a line e^* is drawn so that it crosses e, but no other edge of G, and joins the vertices v^* in the faces adjoining e; these lines are the edges of G^*.

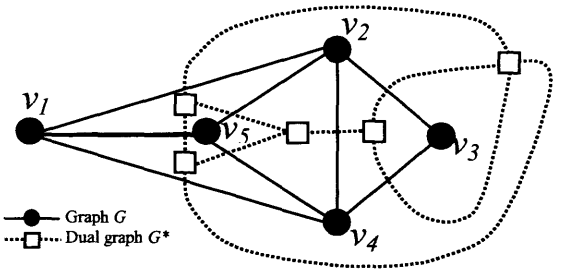


Figure 3.5 - A general example of dual graphs.

It is important to note that any two graphs G^* formed from an initial G in this way must be isomorphic, and that is why G^* should be called "*the* dual of G" instead of "*a* dual of G" (Gibbons, 1989, pg.83; Wilson, 1996, pg. 73; Roberts *et al.*, 2000).

Triangulations are planar graphs whose faces are all triangles. The Delaunay graph for a set of points P can be mentioned as an example, normally denoted *DG(P)*, which is an irregular network of triangles, TIN. This Delaunay graph is the straight-line-dual of the Voronoi diagram, *VG(P)*. From duality, it follows that *DG(P)* is a planar graph. When no four points lie in a circle, then *DG(P)* is a triangulation of P, called the Delaunay triangulation of P, and denoted by *DG(P)* as said.

3.4 Graph search: depth-first search *vs.* breadth-first search

In order to be interpreted, graphs have to be explored and their properties determined by systematically examining each of their vertices and edges. Carrying out this task is cumbersome and equivalent to exploring a maze. Should one be interested in determining some simple graph properties, like computing the degrees of all vertices, this can be easily accomplished by examining each edge. But many other more complex properties of a graph are related to its paths. These can be discovered by moving through the graph, from vertex to vertex along its edges, and by understanding its properties as we go. Indeed, this abstract model is used by most of the graph-processing algorithms (Sedgewick, 2002).

In the literature, two different algorithms are available to accomplish that task: the *depth-first search* (DFS) and the *breadth-first search* (BFS). Although both algorithms traverse the entire graph by systematically visiting all its vertices, the manner in which they operate is different. Briefly, depth-first search goes as further as possible in a given direction "meandering from one node to the next, only backing up to the previous node to try the next possibility whenever it has tried every possibility at a given node" (Sedgewick, 1998); it can be compared to "a single searcher probing unknown territory as deeply as possible" (Sedgewick, 1998). In contrast, breadth-first search "exhausts all the possibilities at one node before moving to the next" (Sedgewick, 1998); it amounts to "an army of searchers fanning out to cover the territory" (Sedgewick, 1998).

A practical result of the traversal process is the generation of a sub-graph that contains all the graph vertices plus the graph edges traversed during the process. Thus, non-traversed edges are not considered. Such a sub-graph is called a spanning tree (*vd.* section 3.3). For illustration purposes, Figure 3.6 represents a simulated map of simple polygons, not classified in any manner, with the respective graph of adjacencies drawn on top of them. Let us choose polygon *2* as the root: the resulting spanning trees, both DFS and BFS, are pictured in Figure 3.7.

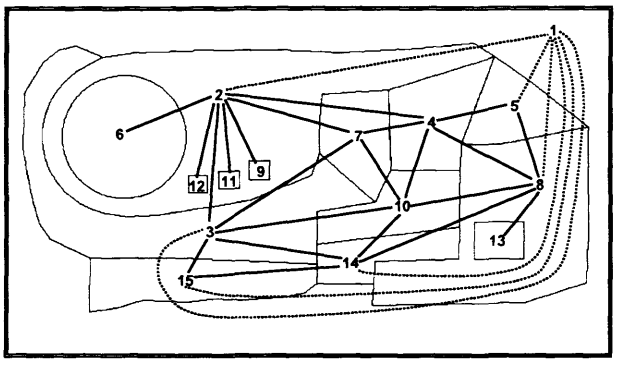


Figure 3.6 - Example of a map of a simulated scene and respective graph of adjacencies.

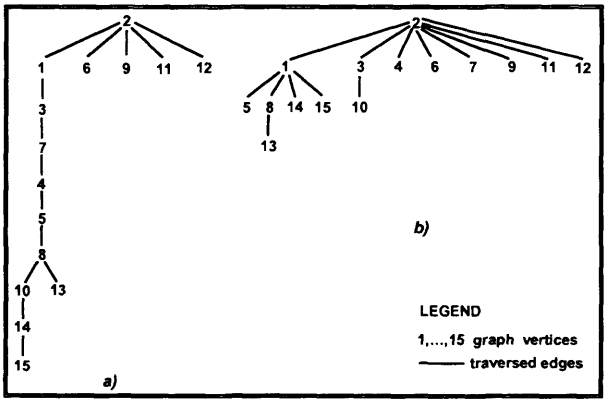


Figure 3.7 - Example of two different traversal trees for the graph in Figure 3.6, both having the same root: a) depth-first search; b) breadth-first search.

3.4.1 Depth-first search

Because DFS and BFS algorithms operate in a slightly different way, though both have similar final aims, each of them is used to accomplish different tasks. According to Sedgwick (1988, 2002), DFS can be used to solve numerous basic graph-processing problems. For instance, this algorithm is mainly used to solve connectivity problems. Furthermore, testing if a graph has a cycle (*vd.* section 3.3) is

a trivial modification of the DFS program; basically, if we encounter an edge pointing to a vertex that has already been visited, then we have a cycle.

From the implementation point of view, DFS can be either recursive or can use an explicit pushdown stack. The recursive implementation of the DFS constitutes the most classic and important of all recursive programs, and operates as follows (Sedgwick, 2002):

- To visit a vertex, the program marks it as having been visited;
- Then, it recursively visits all its adjacent vertices that have not yet been marked.

If the graph is connected, we eventually reach all of the vertices.

As an example, Figure 3.7a) depicts the resulting spanning tree of the DFS traversal process through the graph represented in Figure 3.6 (starting from vertex *2*). This tree is a structural description of the sequence of the traverse function calls, and hence gives us the key to understand how DFS operates (Sedgewick, 2002).

Typically, a DFS tree is deep and thin and carries information about the graph being searched as well as the DFS process (Sedgwick, 2002):

- There exists at least one long path that connects a substantial fraction of the vertices;
- During the search, most vertices have at least one adjacent vertex that has not yet been visited;
- More than one recursive call is rarely done from any vertex;
- The depth of the recursion is proportional to the number of vertices in a graph.

The DFS tree can be augmented to provide a full description of the search dynamics. This can be done by adding external vertices to record the moment when a recursive call was skipped for vertices that had already been visited. Alternatively, another visual representation worth of careful study is the one obtained by simply adding external links representing the non-traversed edges (Sedgwick, 2002).

3.4.2 Breadth-first search

Sedgewick (2002) maintained that, in the implementation of the DFS algorithm, replacing the pushdown stack by a FIFO (*first in, first out*) queue leads us to the BFS

algorithm. As noted above, BFS is analogous to a group of people exploring a maze by fanning out in all directions, as opposed to DFS which is analogous to only one person exploring it.

This algorithm is mainly used to address another class of graph-processing problems related to shortest paths. This is because, as opposed to the DFS, the order in which it takes us through the graph is related to the goal of finding shortest paths. In fact, the distance from each vertex to the start vertex is the key property of interest, the length of the shortest path connecting the two (Sedgwick, 2002). During BFS, "vertices enter and leave the FIFO queue in order of their distance from the start vertex" (Sedgwick, 2002).

Sedgwick (2002) described the implementation of BFS as being "based on maintaining a queue of all edges that connect a visited vertex with an unvisited vertex"; "a loop is put to the start vertex on the queue, and then the following steps are performed until the queue is empty":

- It takes edges from the queue until finding one that points to an unvisited vertex;
- Then, it visits that vertex; puts onto the queue all edges that go from that vertex to unvisited vertices.

Figure 3.7b) depicts an example of a spanning tree resulting from the BFS process carried out for the graph represented in Figure 3.6 (starting from vertex *2*). It "provides a compact description of the dynamic properties of this level order search, corresponding one tree branch to each connected component" (Sedgwick, 2002).

The basic characteristics of the BFS dynamics contrast considerably with those of DFS, and this fact becomes obvious when we observe both traversal trees (*vd.* Figure 3.7). It does not necessarily mean that these characteristics occur in all graphs, however it can be said that, typically, the BFS tree is shallow and broad, and tells us a different set of facts about the graph being searched from those shown by the DFS tree (Sedgwick, 2002):

- There exists a relatively short path connecting each pair of vertices in the graph;
- During the search, most vertices have a number of edges that remain unfollowed because they lead to vertices previously visited.

3.5 How graph theory is applied to spatial analysis

3.5.1 The eXtended Relational Attributed Graph (XRAG)

3.5.1.1 Introduction

To date, standard photo-interpretation skills (*i.e.* human interpretation based on several cues) have been used to analyse remotely-sensed imaged data. However, as is well known, manual interpretation is extremely laborious, time-consuming, and therefore expensive. That is why in 1996 Michael Barnsley and Stuart Barr thought of developing a suitable data processing system to accomplish that task, and came up with what they called the *eXtended Relational Attributed Graph (XRAG)* (Barnsley, 2003).

The authors define XRAG as "a graph-based, structural pattern recognition system that might be used to infer what they call *second-order* thematic information (broad categories of land-use) from *first-order* thematic information (normally, from high spatial resolution remotely-sensed images)" (Barr and Barnsley, 1997; Barnsley and Barr, 1998; Steel *et al.*, 2003). Basically, the input data is a land-cover map (in other words, a map of forms) to which is applied the XRAG to store, analyse and interpret the structural properties of, and relations between, discrete land-cover parcels of the input map, and, as a result, derive a land-use map (in other words, a map of functions). Figure 3.8 roughly summarises this process.

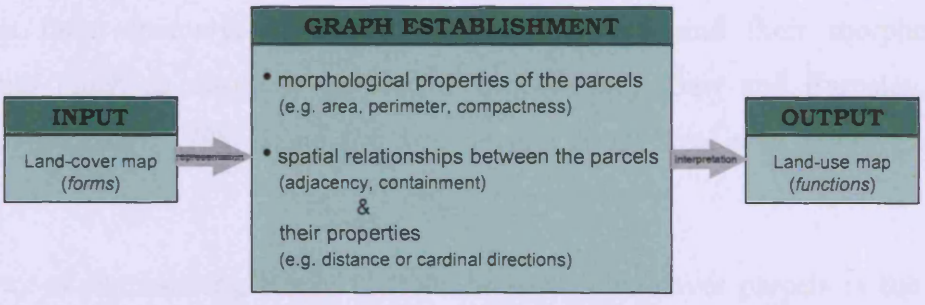


Figure 3.8 – The inference process of second-order thematic information about a scene from first-order thematic information, using XRAG.

3.5.1.2 Description

In conceptual terms, the system uses a graph (in the discrete mathematics sense of the world) to represent the structural relations between the land-cover parcels. Each

one of these parcels is represented by a vertex in the graph, while the spatial relation between two such parcels is represented by an edge connecting them. In Figure 3.9a) there is an example of a very simple urban scene, consisting of some buildings, roads and areas of open space, to which XRAG is applied. As an example, the spatial relations of *adjacency* and *containment* between the land-cover parcels are represented, respectively, in Figure 3.9b) and Figure 3.9c).

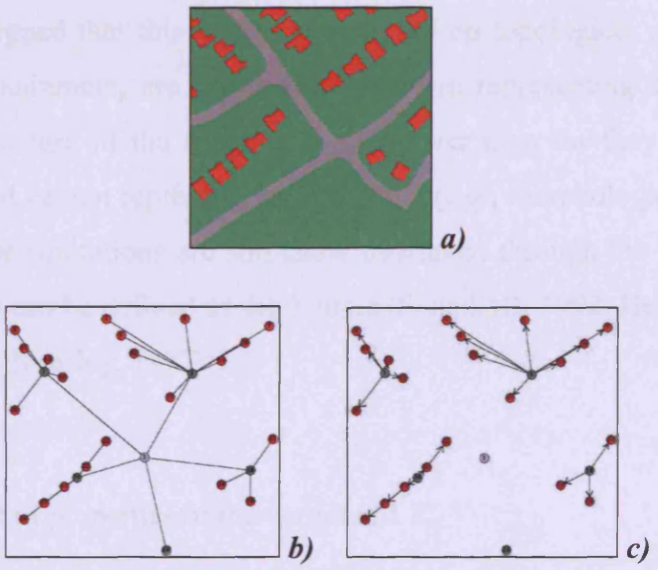


*Figure 3.9 – **a)** A simple urban scene (from Barnsley, 2003); **b)** Graph representation of the spatial relation of ADJACENCY between the land-cover parcels (from Barnsley, 2003); **c)** Graph representation of the spatial relation of CONTAINMENT between the land-cover parcels (after Barnsley, 2003).*

The system is able to represent and analyse the morphological properties of the individual parcels (such as areas, perimeter, shape, etc), various structural relations between them (namely, adjacency and containment) and their morphological properties (such as distance, cardinal directions, etc) (Barr and Barnsley, 1997; Barnsley and Barr, 1998; Bauer and Steinnocher, 2002; Steel *et al.*, 2003; Barnsley, 2003).

One way of representing those relations between land-cover parcels is the use of *graphical pattern representation* techniques (Schalkoff, 1989, 1992, cited in Barr and Barnsley, 1997). This sort of representation is based on graph theory for the representation of the spatial relations.

As explained in section 3.2, a simple labelled graph consists of a finite, non-empty set V of the vertices and a set of edges E, which indicate the existence of a relation

between pairs of vertices in V, and is represented by the equation $G=(V, E)$. In the majority of the cases XRAG was applied to, each member of V corresponds to a unique land-cover parcel, while the existence of an edge between to vertices means that a relation between the corresponding parcels exits.

This kind of graph has already been used in previous studies to represent spatial structure, namely in classified remotely-sensed images (Barr and Barnsley, 1997). However, it is argued that this graphs even based on topological relations, such as adjacency or containment, are somewhat limited in representing the whole spatial and semantic structure of the scene in a land-cover map for they only describe a single relation and cannot represent non-relational (*e.g.*, morphological) properties of the parcels. These limitations are somehow overtaken through the use of attributed graphs, G_a which can be defined as the 3-tuple (Schalkoff, 1992; Heijden, 1994, both cited in Barr and Barnsley, 1997):

$$G_a = (V,E,P) \qquad \textit{(Equation 3.5)}$$

where P is the set of properties of the vertices of V.

Attributed graphs can even be more enhanced by extending the set E and splitting it up into several subsets each of which corresponding to a specific relationship; also by adding P to allow the representation of structural properties (not only of the elements of V but also of those of E). According to Schalkoff (1989, 1992) and Heijden (1994) (both cited in Barr and Barnsley, 1997) attributed graphs of this kind have been extensively used in computer vision for automated image interpretation and image understanding purposes.

The XRAG developed by Barr and Barnsley (1997) is in itself an extension of the basic attributed graph, G_a, to incorporate additional series of sets, and hence the system provides a far more complete, explicit model of the structure of an image which, according to its authors, represents one advantage over simple attributed graphs.

The logical division of the structural relations and properties in XRAG is represented by the 7-tuple (Barr and Barnsley, 1997; Barnsley and Barr, 1998):

$$XRAG = (V,E,EP,I,L,G,C) \qquad \textit{(Equation 3.6)}$$

where,

- V is the set of vertices (or nodes), such that $V \neq \{\}$;
- E is the set of (*extrinsic*) spatial relations between $v \in V$;
- EP is the set of properties associated with the relations in E;
- I is the set of (*intrinsic*) properties relating to $v \in V$;
- L is the set of labels (*interpretations*) assigned to $v \in V$;
- G is the set groups binding $\forall l \in L$ to the context of a scene interpretation; and
- C is the set stating the confidence to which $l \in L \rightarrow v \in V$.

In Equation 3.6 the set V, E and I correspond to V, E and P in the standard attributed graph, G_a of Equation 3.5. The sets EP, L, G and C are the additional ones created as a result of dividing the structural relations and properties of the land-cover parcels into logical groups according to the form of the information they represent.

In basic terms, the XRAG data structure consists of two distinct parts: the *header* and the *body* (Barr and Barnsley, 1997) (Figure 3.10).

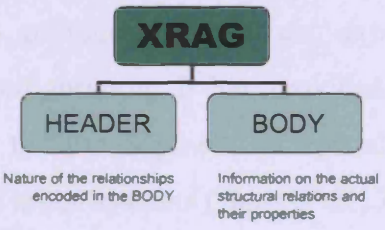


Figure 3.10 – XRAG data structure.

Essentially, the *header* describes the nature of the relations and properties (both of the parcels and of their relations) encoded in the *body*, so that they can be accessed rapidly during data processing. It comprises 13 data fields (3 of *string* type, 4 of *integer* type, 2 of *real* type and 4 *lists*) used to encode information describing *(i)* the files used to initialise the XRAG data structure, *(ii)* the geographical origin of the scene and *(iii)* the number of land-cover parcels stored in the XRAG *body*.

The XRAG *body* encodes information on the actual structural relations and their properties. It comprises a contiguous array, the number of elements of which is equal to the total number of parcels stored. Thus, this array corresponds to the V set of the graph vertices. Each element in the *body*, vertex, contains a series of pointers to other

data types: *Externals* corresponding to the set *E*, *Internals* that corresponds to *I*, *External_Properties* to *EP*, *Labels* to *L* and *Confidence* to *C*.

According to its authors, the XRAG model is similar, in some aspects, to a semantic network (Ballard and Brown, 1982; Schalkoff, 1992; cited in Barr and Barnsley, 1997) which is typically a labelled digraph (*i.e.* a directed graph), which can be used to represent one or more relations between (and properties of) objects, concepts, situations or actions (Schalkoff, 1992, cited in Barr and Barnsley, 1997). Nevertheless, it should be pointed out that there are several differences between the XRAG and a semantic network and the most important is that "the relations and properties represented in XRAG are organized into logical sub-sets on the basis of the form and type of information that they represent". Actually, in a standard semantic network, there is no conceptual difference between an edge that relates to a morphological property (*e.g.* "is larger than") and one that describes a spatial relation (*e.g.* "is contained by") (Barr and Barnsley, 1997).

3.5.2 Space syntax

The space syntax method provides an efficient experimental approach to the understanding of spatial configuration. It has provided since the 1980s important computational support for the development of spatial morphological studies, in particular for the analysis of urban systems (Jiang *et al.*, 2000).

Space syntax also attempts to explain human behaviour and social activities from a spatial configuration point of view (Jiang *et al.*, 2000; Bafna 2003). Thus, space syntax can also be described as "a research program that investigates the relationship between human societies and the space from the perspective of a general theory of the structure of inhabited space in all its diverse forms: buildings, settlements, cities, or even landscapes" (Bafna, 2003). Further to this "social logic of space", Bafna (2003) also stated that the ultimate aim of space syntax research is "to develop strategies of description for configured, inhabited spaces (of buildings, settlements, or built complexes) in such a way that their underlying social logic can be enunciated".

Most space syntax studies deal primarily with issues related to urban patterns, but the method has proved to be relevant in a wide range of studies such as: on the scale of urban design and architecture (Jiang *et al.*, 2000); in crime analysis, and accordingly space syntax theory can be used as a component for actions related to the prevention of crime in urban planning and building design (Jones and Fanek 1997, cited in Jiang *et al.*, 2000); for pedestrian modelling (Hillier *et al.*, 1993, cited in Jiang *et al.*, 2000); traffic pollution control (Penn and Croxford, 1997, cited in Jiang *et al.*, 2000); and way-finding processes (Peponis *et. al.*, 1990, cited in Jiang *et al.*, 2000).

The primary object of analysis within space syntax research, then, is the *configured space*. Having said that a certain space is configured, space syntax theory implicitly assumes that the continuous space can be turned into a connected set of discrete units (large-scale *vs.* small-scale spaces view) (Jiang *et al.*, 2000; Bafna, 2003). There are many advantages in doing so because "different labels can be applied to its individual parts; these parts can then be assigned to different groups, people, or activities; different rules of behaviour and conventions can be associated with different parts of the space; and individual parts of space can be recognized as carrying a specific symbolic or cultural charge" (Bafna, 2003). Indeed, Downs and Stea (1977, cited in Jiang *et al.*, 2000) argued that the cognition of large-scale spaces must build upon the cognition of their small-scale constituent parts.

The computational space syntax model that integrates the small-scale perspective is based on a two-step approach (Jiang *et al.*, 2000): first, the representation of the large-scale space as a finite number of small-scale spaces; second, to link these individual small-scale spaces to form a connectivity graph. Indeed, from the computational point of view, space syntax is based on a graph-oriented representation of the geographical space which adequately captures all the spatial configuration aspects (Jiang *et al.*, 2000; Bafna, 2003).

The spatial decomposition of the urban, *vs.* the building, environment into small-scale components concentrates on free spaces (Jiang *et al.*, 2000). This is because free spaces provide an unique view and hence are fundamental for understanding the configuration of an urban system (Jiang and Claramunt, 1999, cited in Jiang *et al.*, 2000). The distinction between the free spaces and spatial obstacles is generated by the existence of boundaries between streets/rooms and the built environment, *i.e.*

both are interdependent as they share a common physical boundary (Jiang *et al.*, 2000) (*vd.* Figure 3.11a).

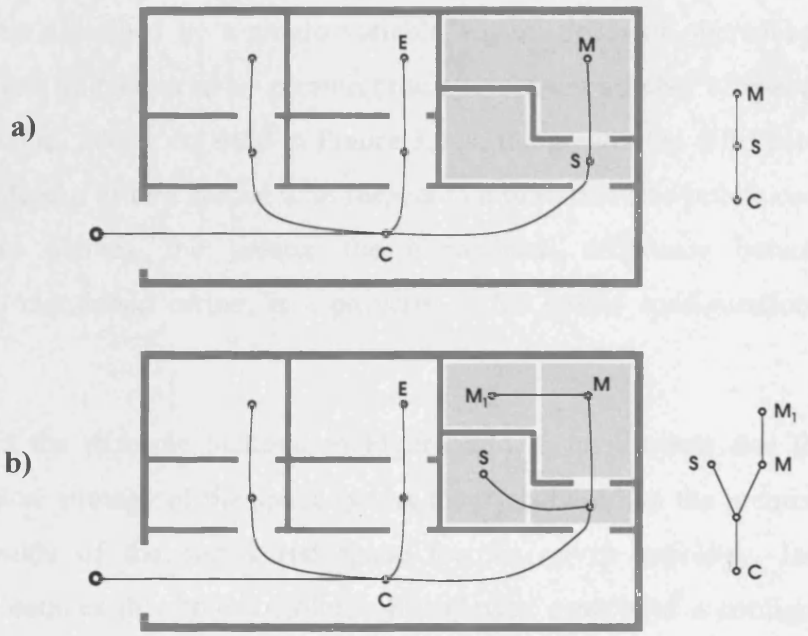


Figure 3.11 – a) Mapping a schematic setting onto a graph; b) mapping non-convex spaces (vertices may not capture the actual configuration of the spatial structure of a setting). (After Bafna, 2003)

Let us consider the example in Figure 3.11a, where the plan of an office corridor is pictured. Given its spatial configuration, it can be said that S is directly accessible from C, whereas M is only accessible from C via S. In other words, the relationship of the manager (M) to his/her administrative assistant (S) is said to be *asymmetrical* with respect to the public corridor (C); on the other hand, the relationship of E and M is *symmetrical* with respect to C. If we represent each room with a vertex and direct access through any two rooms with a link connecting their respective vertices, we could then map the entire building floor by a graph of connectivity. This graph has been the key element of the space syntax spatial analysis. The graph of connectivity supports the understanding of a large-scale space from the perception of its small-scale spaces; this is carried out by computing some important spatial properties, such as how each vertex links to its immediate neighbours, and how each vertex links to every other vertex (Jiang *et al.*, 2000).

Redrawing the graph as shown on the right in Figure 3.11, higher positions within the administrative hierarchy are mapped to upper levels, and hierarchically lower

positions to lower levels. Even the levels of relative privacy of the spaces can be mapped directly onto the levels of the graph. Existing hierarchical relationships as well as patterns of social relationships are two attributes of spatial configuration which can be described by a single variable, *depth*: depth of one space, X, from another, Y, can be measured by counting the intervenient number of spaces between X and Y (Bafna, 2003). As seen in Figure 3.11a, the greater the difference between the relative depths of two spaces with respect to a third one (the public corridor C in the example above), the greater the hierarchical difference between them. Asymmetry, mentioned earlier, is a property of the spatial configuration based on depth.

The point in the example pictured in Figure 3.11 is to illustrate that the general methodological strategy of the space syntax theory is based on the premise that any analytical study of the configured space focuses on its topology. Indeed, this abstraction assumes that "the sociologically relevant aspects of a configured space can be captured at the level of its topological description" (Bafna, 2003).

Given the relevance of the graph in space syntax, the problem of reducing any configured space to an appropriate graph is particularly pertinent and raises another issue, the conversion of a continuous entity (the given spatial unit) into a discrete one. According to various authors (including for instance Jiang *et al.* 2000, and Bafna 2003), there are several space syntax representations which can be applied to a given spatial setting, depending on the degree of linearity of the free space. One simple technique, so called *boundary partitioning*, oriented toward environments in which the free space is non-linear, is to let the partitioning follow the preset physical boundaries, *e.g.* the walls in a building clearly demarcating its rooms (Jiang *et al.*, 2000; Bafna, 2003). However this apparently straightforward procedure may reveal limitations in capturing the main characteristics of a particular setting with non-convex small-scale units. In such a situation, the small-scale units may need to be subdivided into smaller convex subunits (*vd.* Figure 3.11b), so that the graph's vertices capture the actual sociological potential of the spatial structure of the given setting (Bafna, 2003). This technique, so called *convex space partitioning* (or *convex map*), is among the most common means of describing spatial configurations,

especially for the analysis of building plans; a connectivity graph is derived by taking rooms (or convex parts of rooms) as vertices, and door connections as graph's links.

The second space-syntax representation, so called *linear* or *axial map*, is oriented toward environments which are relatively linear and seeks to capture structure of movement within a setting through the alignments of its constituent convex spaces. It is defined as the least number of longest lines. According to how each line intersects other lines, a connectivity graph can be derived taking axial lines as graph's vertices and line intersections as graph's links (Jiang *et al.*, 2000; Bafna, 2003). Axial maps were initially constructed to describe urban areas in which the structure of their street network could be described as a discrete spatial configuration. The underlying intuition is similar to that of convex spaces, based on the notions that (Bafna, 2003): first, the line of sight is a significant organising and unifying device in experience; second, the number of distinct turns on a route are more crucial to the spatial experience than actual distance covered. Thus, the actual distance between two spaces is counted through depth, namely in terms of turns along a path rather than as actual journey length.

Jiang *et al.* (2000) presented a third possible space-syntax representation also oriented to non-linear free space, but with a more precise spatial representation. This representation is based on the notion of *isovist*, which is defined as a visual field that is wholly visible from a single vantage point (basically, the polygon created by delineating the area visible to an observer in that position, most often assumed as having a 360-degree field of vision) (Benedikt, 1979, cited in Jiang *et al.*, 2000). According to this kind of spatial representation, a building plan is partitioned into a finer grid; each cell at the finer level represents a single vantage point and its associate isovist (for more details see Turner and Penn, 1999, cited in Jiang *et al.*, 2000). Then a connectivity graph can be derived depending on how each isovist overlaps each other isovist.

The choice between using, for instance, a convex or an axial map for describing the spatial configuration of a given spatial setting depends on several factors (Bafna, 2003): if the analysis is used to discuss arrangement of programmatic spaces and generative types of buildings, convex map is commonly used; if the focus of analysis

is the understanding of behavioural characteristics of the spatial setting, axial maps are more useful.

For urban morphological analysis, space syntax provides a range of spatial analysis parameters derived from the connectivity graph. The first is, *connectivity*, the most apparent property parameter for morphological analysis (Jiang *et al.*, 2000; Bafna, 2003). Connectivity is defined as "the number of vertices directly linked to each individual vertex in the connectivity graph" (Jiang *et al.*, 2000):

$$C_i = k, \qquad \textit{(Equation 3.7)}$$

where k is the number of vertices directly linked to vertex i.

The second parameter is the *control value*, which is defined as "the parameter which expresses the degree of choice each vertex represents for vertices directly linked to it"; the control value of a vertex i is determined according to the following calculation (Jiang *et al.*, 2000):

$$ctrl_i = \sum_{j=1}^{k} \frac{1}{C_j}, \qquad \textit{(Equation 3.8)}$$

where k is the number of directly linked vertices of a given vertex i, and C_j is the connectivity of the j*th* directly linked vertex.

The depth of a graph's vertex, mentioned earlier, is an important variable for calculating the *integration* of a vertex (Jiang *et al.*, 2000; Bafna 2003). First of all, let d_{ij} be the shortest distance between two vertices i and j in a graph G, then the total depth of vertex i is the sum of distance (Jiang *et al.*, 2000),

$$TD_i = \sum_{j=1}^{n} d_{ij} \qquad \textit{(Equation 3.9)}$$

and accordingly, mean depth is defined by (Jiang *et al.*, 2000),

$$MD_i = \frac{TD_i}{n-1}, \qquad \textit{(Equation 3.10)}$$

where n is the number of vertices of the whole graph G.

Thus, the third parameter, *integration* of a vertex, is by definition "expressed by a value that indicates the degree to which a vertex is integrated or segregated from a system as a whole (*global integration*), or from a partial system consisting of vertices a few steps away (*local integration*)" (Jiang *et al.*, 2000). It can be measured with either *Relative Asymmetry* (RA), or *Real Relative Asymmetry* (RRA) (Jiang *et al.*, 2000; Bafna, 2003):

$$RA_i = \frac{2\times(MD_i - 1)}{n-2} \text{ and } RRA_i = \frac{RD_i}{D_n}, \qquad \textit{(Equation 3.11)}$$

where $D_n = \frac{2n\log_2^{((n+2)/3)-1}+2}{(n-1)(n-2)}$ is the D-value which is intended to provide the standardised value for the integration parameter (for details see Krüger 1989, cited in Jiang *et al.*, 2000).

These parameters can be used to describe both local and global properties of a spatial configuration in the sense of integration or segregation (Jiang *et al.*, 2000; Bafna 2003). Higher integration values of vertices, therefore, indicate that the vertex is less deep on an average from all other vertices, *i. e.* that it is more integrated into the spatial system; in contrast, a vertex has a lower integration value (*i.e.* is more segregated) if the necessary number of intermediate spaces to reach all the other spaces increases (*i.e.* the vertex is deeper). Jiang *et al.* (2000) and Bafna (2003) stated that there is a correlation between these local and global parameters. Such correlation is *intelligibility*, which is used in turn to describe the part-whole relationship within a given spatial configuration.

Bafna (2003) added that several other numbers associated with the graphs have been used in many space syntax studies as well as architectural morphology studies. Those parameters include the number of the graph's rings, or cycles, and the ratio of its vertices to its edges; also, some derivative characteristics such as the ordering of programmatic labels by the integration values of their representative vertices.

Recently space syntax emerged as a new modelling concept in GIS. The relative and absolute views of space have been the two main concepts used for the representation of environmental and urban systems within a GIS: the relative view considers space as a collection of objects, since the objects themselves are the space; the absolute

view considers space as the content of things (Nunes, 1991, cited in Jiang *et al.*, 2000). Space syntax approach provides a different view of space and, according to Jiang *et al.* (2000), the implementation of an analytical space-syntax tool within a GIS provides some important advantages from both the computational and user points of view; furthermore, it promotes both GIS and urban morphology research, and enhances GIS functionality in spatial analysis into the domain of urban morphological.

Jiang *et al.* (2000) developed a prototype, called Axwoman, which is based on the vector data structure of a GIS for the representation of the graph components of the space syntax. Axwoman was implemented as an ESRI's ArcView extension, and basically provides three main functions (*vd.* Figure 3.12): drawing, computation, and analysis. The three functions were implemented using *Avenue* scripts with different interface modules: drawing with view, computation with view and table, and analysis with table and chart. As mentioned above, all these *Avenue* scripts are packed as an extension named Axwoman.avx (Jiang *et al.*, 2000).

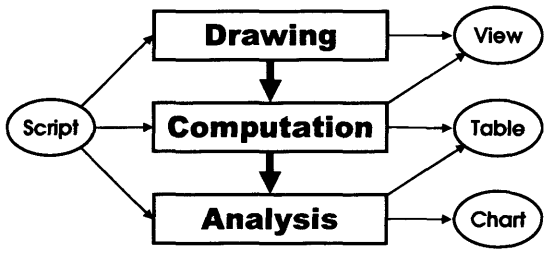


Figure 3.12 – Schematic structure of Axwoman.
(After Jiang et al., *2000)*

According to its authors, once Axwoman is launched a series of morphological analyses can be done: *axial line and polygon drawing* tools allow users to draw axial lines and polygons on a given map of a spatial configuration; *doit* tools calculate various space-syntax parameters, based on a connectivity graph, including connectivity, control value, and local and global integration. Computed results in turn are stored in a table corresponding to the axial map or polygon theme. Users can explore data from different perspectives, or import observed data, such as pedestrian or vehicle flow rates, which in turn can be associated with the integration through regression analysis. In addition, a range of *selection* tools are implemented with different polygon shapes so that local areas can be selected for intelligibility analysis.

Furthermore, a range of analytical components is provided, such as axial maps, polygon maps, tables, charts, and so forth, in order to conduct the morphological analysis discussed in this section (Jiang *et al.*, 2000).

3.5.3 β-Skeletons

3.5.3.1 Introduction

The description and characterisation of the shape and form of planar point sets and their embedded networks has been one important task in the field of computational geometry. Talking about the shape of a point pattern means both its "external" and "internal" shapes.

The simplest geometrical frameworks for describing the "external" shape of a point pattern are the *minimum bounding box* and *minimum bounding circle* (and its generalisation, the *minimum bounding ellipse*). From the viewpoint of descriptive complexity, the next step is to consider the convex polygon of minimum area that contains the entire point set. Different approaches like the *convex hull*, *α-hull* or the *shape hull* of a dot set were developed for this purpose (Toussaint, 1980b, cited in Kirkpatrick and Radke, 1985; Edelsbrunner *et al.*, 1983).

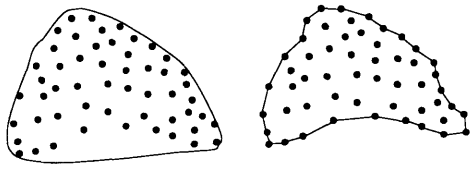


Figure 3.13 – As an example, two different α-hulls of a common point set: a positive and a negative one respectively. (After Edelsbrunner et al., *1983)*

β-skeletons refer to the description of the "internal" shape of a point pattern and is a methodology developed by David Kirkpatrick and John Radke (1985) which basically, and in a few words, is based "on parameterised measures of neighbourliness related to the structure of empty neighbourhoods, with the specific aim of describing the *internal structure* (or *skeleton*, a term used by Toussaint, 1980b, to describe what is fundamentally the same notion) of point sets". For those authors, "the *internal structure* of a planar point set is exhibited by identifying the *essential* points of the set and, among these, joining *essential* neighbours".

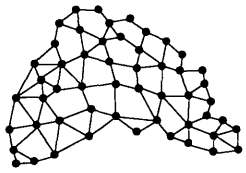


Figure 3.14 – A β-skeleton of the point set showed in Figure 3.13. (After Kirkpatrick and Radke, 1985)

This raises two questions: the notions of "neighbourhood" and "essential points". According to the work surveyed by Toussaint (1980b, cited in Kirkpatrick and Radke, 1985) there are several notions of neighbourhood. We can think of the diameter, minimum bounding box and convex hull as characterising the *global* neighbourhood of a point set (which provides, as said above, a description of the "external" shape of the point set). Analogously, the set of *nearest neighbour pairs* (NNG), *minimum spanning tree* (MST), *relative neighbourhood graph* (RNG), which in turn is a super-graph of the MST and a sub-graph of the GG and DT (Toussaint, 1980a), *Gabriel Graph* (GG) and *Delaunay triangulation* (DT), all describe *local* notions of neighbourhood (which provides a description of the "internal" shape of the point set). Besides that, it is important to emphasise that all those structures mentioned above are graph theoretical techniques used in clustering methods and therefore examples of graph theory applications as well.

Regarding "essential" points, the identification of "essential" extreme points "is intimately linked with the process of determining their interconnections". In practice, "it is often possible to assert that certain points in a dataset are more "essential" than others by employing non-geometrical knowledge of the data" (both quotations from Kirkpatrick and Radke, 1985).

Generally, NNC $\subseteq$ MST $\subseteq$ RNG $\subseteq$ GG $\subseteq$ DT which provides a discrete hierarchy of internal shape descriptors (Toussaint, 1980a; Kirkpatrick and Radke, 1985). Kirkpatrick and Radke sought the unification and generalisation of the above hierarchy based only on the simple idea of "progressively larger excluded neighbourhoods" and came up with a more detailed description of the mentioned hierarchy of internal shapes called β-skeletons, which, as said above, are based on a parameterised notion of neighbourhood. According to these authors, such a hierarchy serves two purposes: "it allows us to visualize a spectrum of internal shapes of various edge densities. The entirety of this spectrum, including, in particular, the

transitions between adjacent structures, provides an added dimension for the representation of the structure. The second advantage is that it serves as a kind of benchmark with the aid of which empirical networks can be analyzed and, to some extent, compared".

3.5.3.2 Description

Let us consider a pair of points x and y. The continuous family of neighbourhoods is indexed by a single positive real valued parameter β. $N(x,y,\beta)$ denotes the neighbourhood with index β. Moreover, it is assumed that

$$N(x, y, \beta_1) \subseteq N(x, y, \beta_2) \text{ for all } 0 \leq \beta_1 \leq \beta_2 \qquad \text{(Equation 3.12)}$$

$N(x, y, 0)$ is taken to be some appropriately small neighbourhood of the pair x, y, whose emptiness defines a necessary condition for the pair to be considered neighbourly. Similarly, as β approaches ∞, $N(x,y,\beta)$ describes some appropriately large neighbourhood of x and y whose emptiness defines a sufficient condition for the pair to be neighbourly (equivalently, whose non-emptiness defines a necessary condition for the pair to be non-neighbourly) (Urquhart, 1980; Kirkpatrick and Radke, 1985).

Assuming all this, we can say that two points x and y are *β-neighbours* in the set S if $N(x, y, \beta)$ contains no point of S, other than x or y, in its interior. Therefore, *β-skeleton* of a point set S is defined as "the set of edges joining *β-neighbours* in S". We say that the *β-value* of the pair (x, y) is the largest β such that x and y are *β-neighbours*. This permits a more precise notion of neighbourhood, namely, x and y are *(β_1, β_2)-neighbours* if $\beta_1 \leq \beta\text{-value}(x, y) \leq \beta_2$. The *$\beta$-skeleton spectrum* of S is the complete graph on the vertex set S each edge of which is labelled with the *β-value* of the corresponding pair. As its name suggests, the β-skeleton spectrum of S encodes the entire family of β-skeletons for the pair of the set S (Kirkpatrick and Radke, 1985).

Another question that naturally arises from all this is the notion of *neighbourliness*. Kirkpatrick and Radke (1985) comment that the assumption of indexed neighbourhood based upon distances between points would seem quite straightforward. This is because of the intuitive idea that points which are close to

one another are neighbourly. According to these authors, this intuition is fairly close to the truth in several applications, like clustering; nevertheless, only relying on absolute distances as means to measure neighbourliness may lead us to unrealistic results. This is because, when broadly applied, neighbourliness ought to be scale dependent. Moreover, although scale independence could be provided by an appropriate normalization, there remain problems created by variation in density or internal scale (Kirkpatrick and Radke, 1985). To exemplify, these authors suggest thinking of points that are clustered. Then, a typical β-skeleton based on such a definition would be highly connected and non-planar in some regions and perhaps disconnected as a whole. While it is true that this reflects something about the point set structure, at the same time it ignores other important aspects of internal shape, such as the relative positions of nearby points.

Therefore, Kirkpatrick and Radke (1985) de-emphasise the role of distance in defining neighbourliness and suggest:

> ***(i)*** The significance of absolute distance is application dependent; thus, it should be combined with other measures of neighbourliness as dictated by the application at hand.
> ***(ii)*** All the geometrical structures previously used for internal shape description are scale dependent; in keeping with them, the authors seek to take full advantage of the geometrical properties of the data, recognizing the importance of such intuitive notions as the "interference" of points.

3.5.3.3 Examples of applications

Kirkpatrick and Radke (1985) proposed the use of β-skeletons as descriptors of the internal shape of point patterns. These authors argued that the principle advantage of the family of β-skeletons over such fixed structures as the MST, RNG or GG is its versatility. The respective β-skeleton of a point set will vary from a very sparse (typically empty) graph to a very dense (typically complete) graph, depending on the choice of β. The authors sustained that "the combination of this variety with the knowledge of the incremental transitions that fill out this spectrum, provides an added dimension in the characterization of shape that cannot be captured by any single structure". In addition, applications on point sets with a very regular internal structure showed that "the respective set of distinct β-skeletons is much smaller than

for random points sets of comparable size, and the transitions between those β-skeletons are more striking".

According to Radke (1982, cited in Kirkpatrick and Radke, 1985), probably the most interesting application of this framework is to the analysis of edge patterns in empirical networks. Indeed, it is the application for which the concept of β-skeletons was first developed. In this case, the β-skeleton spectrum serves as a continuous benchmark, against which individual links of real networks can be compared.

Some interesting examples of applications described in Kirkpatrick and Radke (1985) cover both artificially generated and empirical networks. One of them shows the success of the β-skeletons methodology in revealing the non-homogeneity in general of the artificially constructed networks. Other examples of the application of the same approach to the analysis of both planar and non-planar empirical networks, test and show its ability to detect significant patterns in these kinds of networks allowing, for instance, the inference of reasons explaining both the existence of *least expected links* (typically with low β-values) and *missing links* (typically with high β-values, *a priori* the ones whose appearance is most expected).

3.5.4 Urban network analysis

The network approach has been widely used in urban studies to represent sets of roads, streets, pipes, aqueducts, power lines, etc., which are interconnected. Urban networks are mathematically modulated by graphs. Indeed, most of the transportation problems, if not all, are based in a quite simple and intuitive representation of networks in which vertices represent road intersections (or settlements), and edges represent roads (or lines of relationship) (Laurini and Thompson, 1992; Porta *et al.*, 2006). In turn, as has been mentioned in this chapter, network analysis procedures are commonly based on graph algorithms (Laurini and Thompson, 1992; Sedgewick, 1988, 1998).

The determination of optimal routes in a given network, typically shortest paths either in terms of total length or time, is a common problem when analysing road networks. From the computational perspective, the applications developed so far in order to solve such problems are strongly based on well acknowledged graph

algorithms, such as BFS algorithm (Sedgewick, 2002) discussed in section 3.4. Moreover, facility location and routing problems are frequently multiobjective in nature (Coutinho-Rodrigues *et al.*, 1994, 1997). In other words this means that, given an urban network, the selection of a route from an origin to a destination is influenced by several objectives.

The most complex aspect of the need to deal with more than one objective is the fact that these objectives are often in conflict (Coutinho-Rodrigues *et al.*, 1994). When this occurs, no single route is optimal for all objectives; for instance, the optimal route that minimizes the total cost may not to be the one that minimizes the delivery time. In such cases, a logistics manager would have to analyse the tradeoffs among the various objectives that exist among the various routes, and select the option that best suits his/her preferences. Ideally, the decision-maker wants to consider only those solutions that are *efficient* (also referred to as *non-inferior*, *non-dominated*, or *Pareto-optimal*) (Cohon, 1978, cited in Rodrigues *et al.*, 1997). An *efficient* solution is defined as the one in which the improvement in one of the objectives implies a degradation in one or more of the other objectives (Coutinho-Rodrigues *et al.*, 1994). Hansen (1980, cited in Rodrigues *et al.*, 1997) argued that the number of efficient solutions may increase exponentially with size problem even in the most basic multiobjective routing problem, such as the bicriterion shortest path problem.

As an example, Figure 3.15 depicts a hypothetical efficient frontier for a given road network. Lets us consider a bicriterion problem, assuming that objective Z_1 is to minimize cost, and objective Z_2 is to minimize total travel time. Points A to E represent the objective function values for five possible routes.

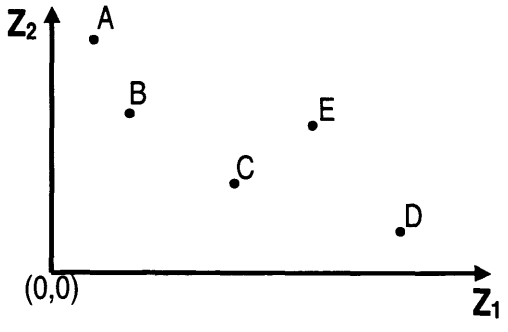


Figure 3.15 – Route-solutions (represented by points) in a two-objective space. (From Coutinho-Rodrigues et al., *1994)*

Routes A, B, C, and D are members of the efficient frontier for this particular problem. Solution A is the route that minimizes transportation cost and solution D is the one that minimizes total travel time. In turn, routes B and C represent intermediate or compromise solutions. Finally, route E represents an inferior solution and is not a member of the efficient frontier as route C is better than E in both objectives, and should therefore be preferred over E. A visual interpretation of the graph depicted in Figure 3.15 shows that, given the efficient frontier, a relatively large reduction in travelling time for a relatively small increase in total cost seems possible to obtain just by moving from route A to route B; however, when moving from route C to D, one must incur a relatively large increase in cost for a relatively modest reduction in travel time. This shows that the choice of the actual route depends on one hand on the magnitude of the objective function values, and on the other hand on the logistics decision-maker's preferences (Coutinho-Rodrigues *et al.*, 1994).

As Cohon (1978), and Teghem and Kunsch (1986) have argued (both cited in Coutinho-Rodrigues *et al.*, 1997), as the number of objectives increases, the analysis of the tradeoffs among the various objectives and among the various efficient alternatives becomes more cumbersome. In addition, individual routes may require considerable computation time to generate. As a consequence, it is generally impractical to generate the entire efficient frontier for most realistic logistics routing problem (Coutinho-Rodrigues *et al.*, 1994).

Coutinho-Rodrigues, Current, Clímaco and Ratick (1994 and 1997) have demonstrated that it is possible to design methods to mitigate the effect of the difficulties mentioned above, and thus to enable decision-makers to identify the inherent tradeoffs in multiobjective problems. As an example, the PC-based interactive spatial decision-support system (ISDSS) for multiobjective location-routing problems (Coutinho-Rodrigues *et al.*, 1997) is briefly reviewed in this section.

ISDSS was initially developed within the context of the transportation of hazardous materials (hazmat) issues. As far as hazmat problems are concerned, the logistic components to be considered are (Coutinho-Rodrigues *et al.*, 1997): the location of

processing facilities; the selection of transportation routes between hazmat sources and these facilities; and the quantities of hazmat shipped over these various routes.

The goals of ISDSS are (Coutinho-Rodrigues *et al.*, 1997): first, to reduce the number of efficient solutions that need to be generated; second, to facilitate the comparison of alternative solutions by the system-generated graphics; and finally, to facilitate editing of network data. The first goal was achieved by a graphical display technique called *BAGAL* (best against least). The other two goals were achieved by techniques previously developed by the authors and others. ISDSS basically is demonstrated by an adaptation of the five-objective hazmat problem (REEM) previously presented by Current and Ratick (1995). REEM is a location-routing problem that incorporates objectives related to cost, risk, and the spatial distribution of the risk imposed. Originally, REEM assumes that wastes cannot be transported through a generating or facility vertex of the network (Current and Ratick, 1995). In ISDSS the model proposed received some modifications in order to allow transportation through generating or facility vertices without needing to modify the initial network (Coutinho-Rodrigues *et al.*, 1997).

In order to test ISDSS, this was first applied to an extract of the New York road network. ISDSS generates graphics that present solutions in: *decision space*; and *objective function space*.

Decision space are maps of the solution that show which facilities are selected, which transportation arcs are utilised, and what the relative flows are on the various arcs (*vd.* Figure 3.16).

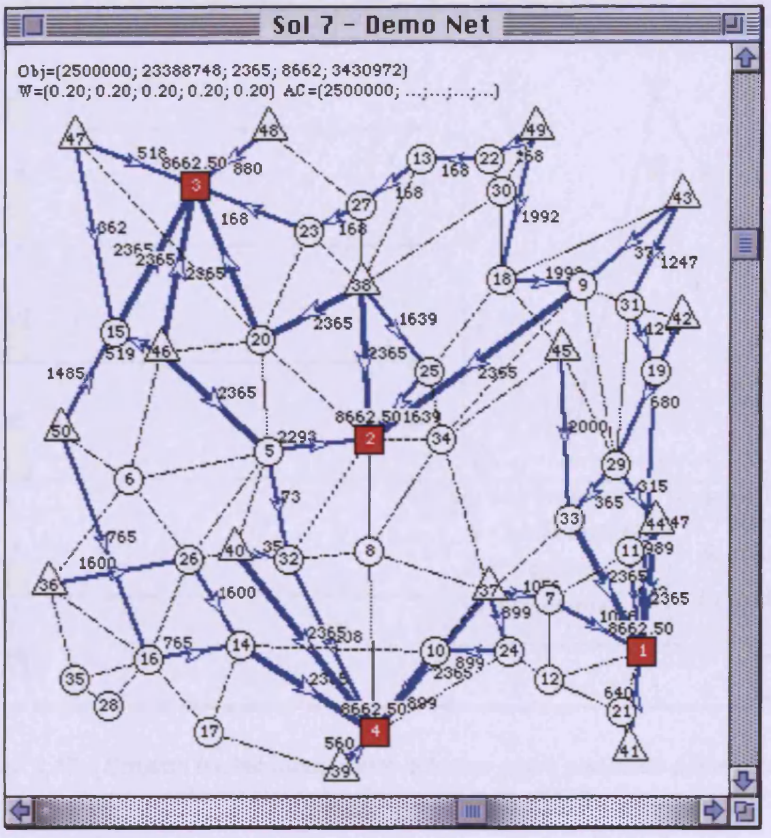


Figure 3.16 – Graphical representation of solutions (for illustrative purposes, solution 7) in decision space.
(From Coutinho-Rodrigues et al.*, 1997)*

Objective function space graphics (*vd.* Figure 3.17) employ one or more of the following graphical techniques that can be selected by the decision-maker: bar graphs, Value Paths (Schilling *et al.*, 1981, cited in Coutinho-Rodrigues *et al.*, 1997), Spider Webs (Kasanen *et al.*, 1991, cited in Coutinho-Rodrigues *et al.*, 1997), and GRADS (Klimberg, 1992, cited in Coutinho-Rodrigues *et al.*, 1997). These graphics are used to compare solutions.

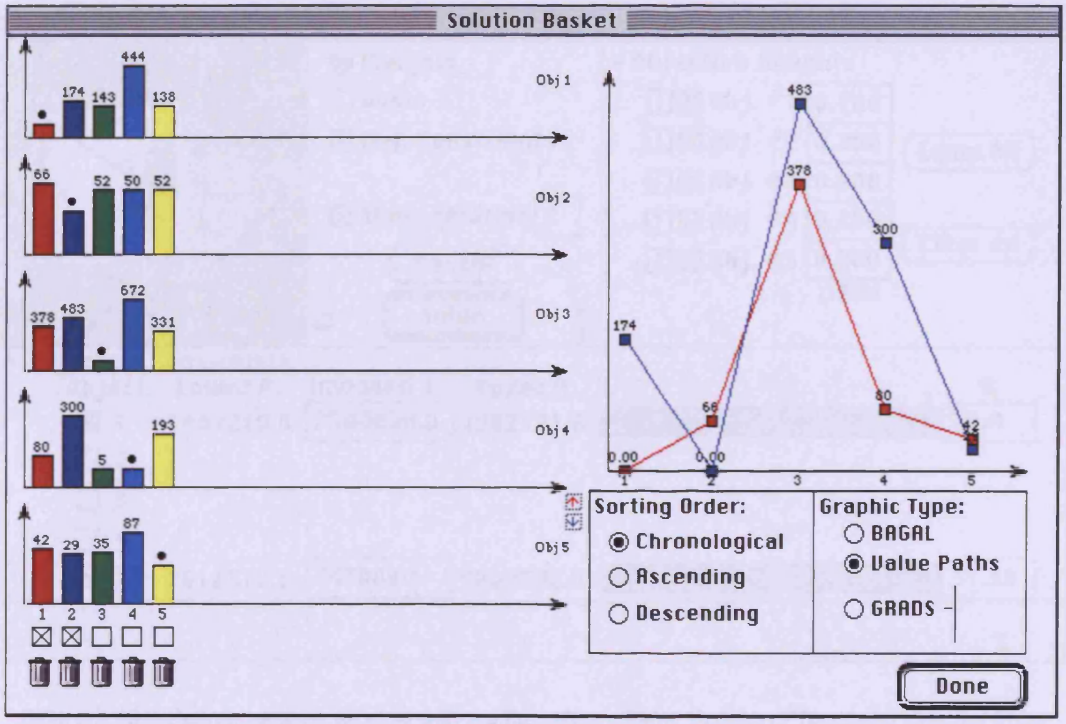


Figure 3.17 – Solution basket window with columns graph and value paths display. (From Coutinho-Rodrigues et al.*, 1997)*

In order to direct the search for new efficient solutions, the weighting method, the constraint method, goal programming, or a combination of these techniques may be employed by the decision-maker. Objective function weights or goal priorities are entered via a dialogue box. Objective function goals or constraints are entered via mouse-controlled slide bars. The effects of these latter inputs on the feasible region in objective space are shown by the BAGAL graphic. Once the decision-maker input has been determined, ISDSS modifies the mathematical formulation of the problem and a new solution is generated, which is added to the solution basket (*vd.* Figure 3.18).

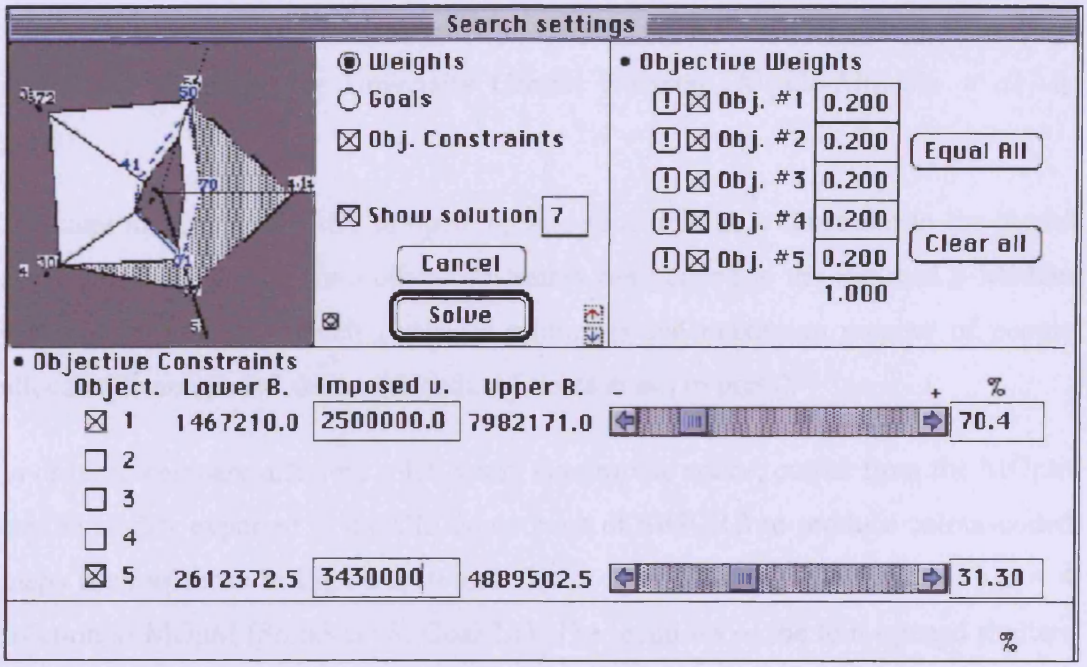


Figure 3.18 – Search settings window with additional constraint. Solution 7 has been directed to be shown; this solution had constraints on Objectives 1 and 5, and equal weights on the objectives. (From Coutinho-Rodrigues et al., *1997)*

As mentioned earlier, ISDSS was presented by its authors within the framework of a hazmat problem. However, the system's basic components can be modified to analyse any multiobjective location, routing, or location-routing problem.

Another graph-based multiobjective approach worth of being mentioned is the SIGURB (Alçada-Almeida *et al.*, in press). SIGURB is a web-based, multicriterion, decision support system, designed to evaluate aspects of the emergency evacuation planning. The system was tested in the Coimbra city downtown (Portugal) which is characterised by a typical medieval urban pattern, consisting of a labyrinth of narrow streets and ancient wooden buildings.

SIGURB was designed to assist planners in determining the number and location of evacuation shelters (or assembly points), and the paths that people should take from their buildings to their assigned shelters, for instance in case of fire. Nine potential shelter sites were identified in the given area. An important component of the system is a multiobjective p-Median model (MOpM) with four objectives: the minimisation of total travel distance for people to reach their "primary shelter" (the first identified shelter by the system); the minimisation of total risk of the "primary path" in case of being blocked due to the fire; the minimisation of the fire risk at the shelters; and the

minimisation of the total time required to rescue people, when necessary, from their evacuation shelter to the University Central Hospital (Alçada-Almeida *et al.*, in press).

The total number of shelters to open, p, was structured as a constraint in the model and varies from 2 to 7. Two other constraints were added to the standard p-Median problem formulation, which impose a minimum and maximum number of people allocated to an opened shelter (Alçada-Almeida *et al.*, in press).

In order to compare different solutions in *geographic space*, output from the MOpM can be readily exported to the GIS component of SIGURB to produce colour-coded maps such as those in Figure 3.19 and Figure 3.20. Figure 3.19 represents a $p = 4$ solution to MOpM (Solution #8; Goal L1). The locations of the four opened shelters are identified with different coloured circles. Buildings assigned to a certain shelter are mapped with that shelter's "colour". These maps are overlaid over aerial photography. Buildings that are not colour-coded in Figure 3.19 are not in the case study area, and white polygons represent empty buildings.

Figure 3.19 – Facility location and building assignment for p=4 (Solution #8, Goal L1). (From Alçada-Almeida et al., *in press)*

According to the USFA (http://www.usfa.dhs.gov/safety/escape), a sound escape plan will greatly reduce fire death and "... the best plans have two ways to get

out...". Consequently, the system identifies two possible evacuation paths for each building, in case the first identified becomes impassable due to the fire (*vd.* Figure 3.20). These two paths are as dissimilar as possible (Alçada-Almeida *et al.*, in press).

Figure 3.20 – Example of Primary and Secondary paths for two given buildings. (From Alçada-Almeida et al.*, in press)*

Figure 3.20 shows the primary and secondary evacuation paths for two sectors marked with a black dot. Buildings are mapped in the same colour as that of the shelter that the building is assigned to. The primary path from a building is displayed with a line in the same colour as that of the shelter that the concerning building is assigned to. The secondary path is indicated with a line in the same colour as that of the alternative shelter.

Two different formulations of the problem were analysed: a night and a day versions of the problem. This was due to the fact that Coimbra downtown is a service-based area with a considerable variation of its population (in number and location) during a 24-hour day; thus, two different planning realities.

The MOpM was solved within a framework of a web-based decision support system that includes a high level interface to mathematically structure the model instances – an algorithm server – and a linkage to a web-based GIS. Various graphical

techniques allow users to analyse non-dominated solutions already generated, and to determinate additional solutions that may be of interest. This was done in *geographic space* via the GIS and in *solution space* via various graphical techniques (similar to those of ISDSS mentioned above).

Although SIGURB was presented in the context of the need for evacuation during major fires, the system can be applied to other emergency situations such as earthquakes, floods, and terrorist attacks.

3.5.5 Built-form connectivity

The built-form connectivity attempts to explain the spatial organisation of an urban system on the basis of its distribution of buildings. It focuses on both the intrinsic spatial relationships of buildings between one another and also on their external relationships with other urban features, such as the road network (Barr and Barnsley, 2004).

Traditionally, urban system connectivity research has concentrated overall on an analysis of the spatial topological organisation of *built forms* (Barr and Barnsley, 2004). A key example is the work of Krüger (1979a, 1979b, 1980, 1981a, 1981b), which is briefly reviewed in this section.

In Krüger's work, *built forms* are defined as "planar edge-connected set representing external walls and partitions (party walls) of buildings on the ground and can be represented by a particular combination of three types of mathematical graph". These three types of graph represent the connectivity relations: amongst the built forms, between the built forms and the surrounding external environment, and between the built forms and the channel network (Krüger, 1979a).

Krüger's starting point is based on both the notions of *homomorphism* and *isomorphism* in the relationship between buildings and built forms[2]. In his model, the

[2] *i.e.*, mappings that relate *many-to-one* and *one-to-one* respectively. In fact, different built-form arrays could be represented by the same graphs of universes U_1 and U_2 (*vd.* Figure 3.22). Although they could be isomorphic (*vd.* Appendix A) within each universe, two sets of arrays may have the same graph representation in one system but different ones in another (Krüger, 1979b).

author defined built forms as "quasi-mathematical models" and used a graph-theoretic representation in order to express how buildings are connected and packed over an area of land: buildings are represented buy points called built forms; external walls and partitions (party walls between buildings) by lines. The connected sub-graphs consisting of built forms and partitions are called arrays of built forms (Krüger, 1979a). According to the author, this constitutes a simplified view of the built-form subsystem that, together with the road channel, gives rise to a graph that models an urban system.

In Krüger's urban system model, the graph components are stratified by a structural tree which consists of different levels of disaggregation (*vd.* Figure 3.21). The whole urban graph system is at the level of greatest aggregation. The first level of disaggregation is divided into two different elements: the *built-form galaxy* and the *channel network*. The former represents all built forms within places, the latter relates to all kinds of links and vertices that represent some physical form of transportation between places.

The built-form galaxy is subdivided into *built-form constellations* (*i.e.* sets of built forms surrounded by part of the channel network). Built-form constellations are subdivided into *built-form arrays* (*i.e.* sets of independent connected components within a constellation of built forms). Last, the *built-form arrays* consist of one or more connected built form units.

In turn, the channel network is subdivided by type (into road, rail and river networks), and each of these is then subdivided into *blocks* (sub-graphs with a cycle structure) and *cul-de-sacs* (sub-graphs without a cycle structure).

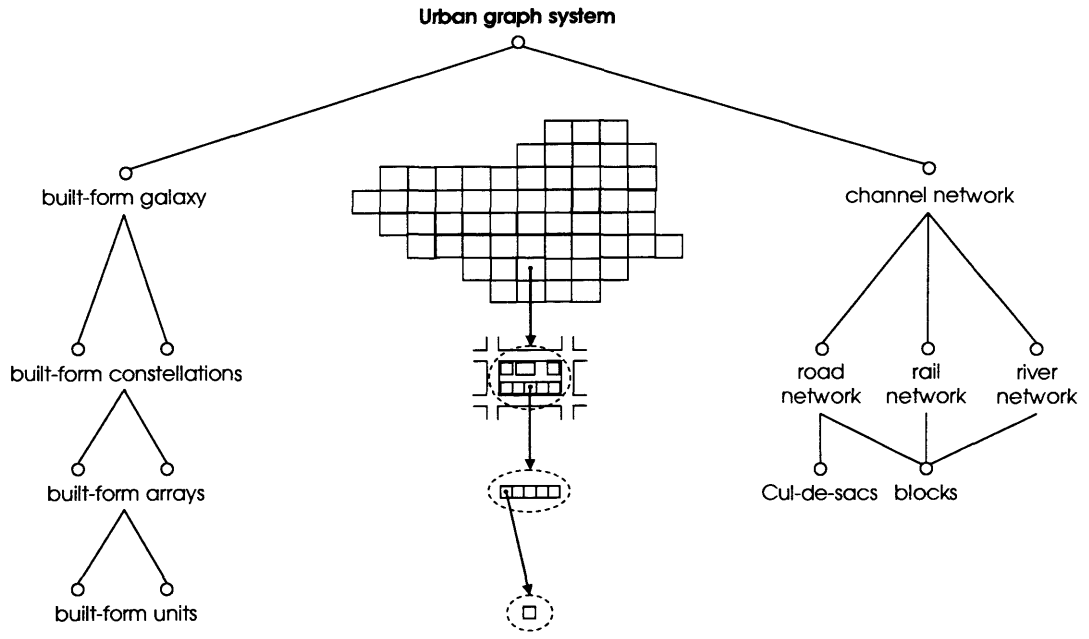


Figure 3.21 – Conceptual hierarchical levels of resolution of Krüger's urban graph system. (After Krüger, 1979a)

In terms of graph representation, three different representations were proposed by Krüger (1979a): *graphs of type 1* – the points represent the centres of the interiors of the polygonal planar edge-connected sets (built forms), and the lines represent the common side walls between edge-connected sets; *graphs of type 2* – the points represent the centres of the interiors of the built forms and the boundary neighbourhood centres adjacent to each external wall, and the lines represent the external walls; *graphs of type 3* – the points represent the centres of the interiors of the built forms and the nearest points on the channel network, and the lines represent the access route to each built form from the channel network.

A graph G_n, of type n, consists of different elements, points (graph vertices, V_n), lines (graph edges, E_n) and independent connected components (C_n), which are structured in a particular way. The universe U_n of all graphs of type n of the urban graph system is the set of all graph elements (vertices, edges, and components) of those graphs (Krüger, 1979a).

For universe U_1, each facet (the interior of a built form) defines a graph vertex, each partition defines a link, and each built-form array a component (*vd.* Figure 3.22). For universe U_2, each neighbourhood centre defines a graph vertex which, when the graph representation is superimposed on the built-form representation, is external to

the appropriate built-form array; all other vertices are defined by facets, as in the case of U_1. The links are defined by the external walls; however, the components in U_2 are not defined by built-from arrays, which in general define only sub-graphs. For U_3, the street connection points and facets define vertices and the access routes define links in the same way as for U_2 (Krüger, 1979a). Krüger (1979b) also presented two other universes, U_4 (to describe the street network), and U_5 (to describe the relationship between city blocks sharing the same streets).

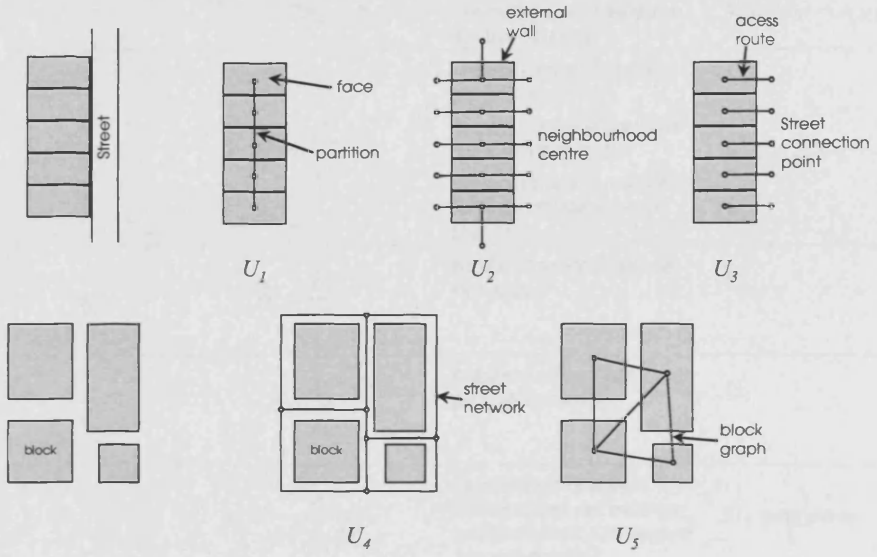


Figure 3.22 – The five universes U_n, $n = 1,\ldots,5$, *used to describe the structure of built forms and their urban settings:* U_1 *describes the partition (party-wall) condition;* U_2 *describes the external-wall condition;* U_3 *describes the connections of built forms to the street system;* U_4 *describes the street network; and* U_5 *describes the relationship between city blocks sharing the same streets. (After Krüger, 1979a, 1979b)*

In order to provide a numerical scale for the urban graph's properties, Krüger (1979a) defined twelve measurements representing either the connectivity amongst the elements of the built-form subsystem or of the spatial relation of adjacency between built forms and their external environment. These measurements are sets of ordered numerical symbols which are based on the sum of graph vertices, edges, and independent connected components. Table 3.1 summarises the twelve graph-theoretic measurements: the first, μ, is a global measurement of structure; α through to γ are connectivity measurements for graph properties; η through to ψ are connectivity measurements for entire graphs (α through to ψ are applied to an individual universe); ξ through to ζ are adjacency measurements, which are ratio

measurements between universes at successive levels in the urban hierarchy (Krüger, 1979a).

Measurement	Equation	Description / Interpretation	Application
μ	$\mu = E_n - V_n + C$	Number of independent cycles in a graph	U_n, $n = 1,...,5$
α	$\alpha_1 = \frac{E_1 - V_1 + C_1}{V_1 - 2C_1}$	Ratio between the observed and maximum possible number of cycles:	
		A measure of the degree of "packing" in built forms;	U_1
	$\alpha_n = \frac{E_n - V_n + C_n}{2V_n - 5}$	A measure of the "connectedness" of the street and block systems	U_4, U_5
β	$\beta_n = \frac{E_n}{V_n}$	Average number of partitions per built form (n=1); Average number of road links per intersection (n=4); Average number of road links forming a boundary to each block (n=5).	U_1, U_4, U_5
δ	$\delta_1 = \frac{E_1}{C_1}$	Number of partitions per built-form array	U_1
ρ	$\rho_1 = \frac{V_1}{C_1}$	Average number of built forms per built-form array	U_1
γ	$\gamma_1 = \frac{E_1}{2V_1 - 3C_1}$	Ratio between observed number of lines and maximum possible number: A measure of "connectedness".	U_1 outer planar
	$\gamma_n = \frac{E_n}{3V_n - 6}$		U_4, U_5 planar
η	$\eta_1 = \frac{E_1 + V_1 + C_1}{E_1}$	Proportional to the sum of the number of vertices and components	U_1
θ	$\theta_1 = \frac{E_1 + V_1 + C_1}{V_1}$	Proportional to the sum of the number of edges and components	U_1
ψ	$\psi_1 = \frac{E_1 + V_1 + C_1}{C_1}$	Proportional to the sum of the number of vertices and edges	U_1
ξ	$\xi = \frac{E_2}{V_1}$	Perimeter: average number of external walls per built form	$U_1 \cup U_2$
π	$\pi = \frac{E_2}{E_1}$	Shape: ratio of the number of external walls to the number of partitions	$U_1 \cup U_2$
ζ	$\zeta = \frac{E_2}{C_1}$	Compactness: average number of external walls per built form array	$U_1 \cup U_2$

Table 3.1 – Krüger's (1979a, 1979b) graph-theoretic measurements of built-form connectivity (V ≡ number of vertices; E ≡ number of edges; C ≡ number of components; n refers to the n^{th} graph universe, n=1,...5).

The measurements summarised in Table 3.1 were used to investigate the structural organisation of both simulated and real urban systems (namely, the ninety-five kilometre-square cells used to record the urban spatial structure of the city of Reading, UK), and also to study the relations between connectivity and adjacency measurements. Correlation analysis was used to determine the degree of redundancy and complementarity between the connectivity and adjacency measurements. For a simulated dataset, which had the same number of built-form units but different spatial topological configuration, clear differences were found in the majority of the measurements (*vd.* Krüger, 1979b, for more details). However, in the case of the urban system of Reading, only five measurements (three connectivity, and two adjacency measurements respectively) were found to be statistically independent: δ, γ, η, ξ and ζ (Krüger, 1979b). These five measurements are said to be complementary in the sense that none is significantly related to another.

Krüger (1980) studied the relationship between the five remaining connectivity and adjacency measurements mentioned above and the urban spatial structure. This was done by means of multivariate statistical techniques at two levels of resolution and spatial aggregation (individual *cells vs. regions* comprising more than one cell) for the kilometre-square cells used to record the city of Reading. The experiments carried out let the author to conclude that the perimeter measurement (ξ) should be discarded as it has no functional relationship with the urban spatial structure; only the four remaining measurements are interdependent with it. After having validated these measurements with the urban spatial structure, Krüger also concluded that they could be used as indices of built-form development (Krüger, 1980).

Krüger (1980) argued that considering the remaining built-form connectivity and adjacency measurements left as indices of built-form development, it would be of interest to generate them from the urban spatial structure and also from some elements of the urban graph. Therefore, an attempt was made to generate one of those measurements, the δ measurement, by modelling the distribution of partitions (*i.e.* party walls between buildings) and components (*i.e.* built-form arrays) for Reading's subsystem, as functions of the distribution of built forms (Krüger, 1981a).

Krüger proposed equilibrium to exist between, on one hand, the supply of built forms and the "demand" made by partitions and, on the other, between the supply of built forms and partitions and the "demand" made by arrays. It was shown that the relationship between partitions and built-form arrays has the form of a power distribution which expresses the equilibrium mentioned above (Krüger, 1981a).

Krüger (1981b) carried out further research in this domain concentrating on the disaggregation of built forms by type, *i.e.* into detached, semidetached, terraced and connected built forms, for the same spatial urban system of Reading. Such disaggregation revealed to be a nonindependent event: the probability of having less connected arrays in a given zone is inversely related to the probability of having more connected arrays in the same cell (Krüger, 1981b). To conclude this investigation, Krüger noted that connectivity properties of the built-form system are neither random nor deterministic but probabilistic. According to him, the main advantage of this approach over the more traditional ways of looking at built-form problems is that predictions are made under conditions of uncertainty (Krüger, 1981b).

Regression analysis of Krüger's graph-theoretic measurements for the city of Reading, in terms of observed values and those for random built-form configurations, lead Krüger (1979b) to conclude that this particular city exhibited a spatially organised built-form structure. In addition, the perimeter measurement (ξ) and the compactness measurement (ζ) indicated that the constellations defined by the road network embedded areas of distinct urban land-use (Krüger, 1979b).

Barr and Barnsley (2004) noted, however, that if the ultimate goal is land-use classification, certain modifications to Krüger's model are required due to the region-based representation concept. In fact, those authors questioned some of the results obtained by Krüger (1979b) because it was not guaranteed that each constellation is considered as an individual (*i.e.* a single $1km$ cell may well contain more than one constellation). Moreover, the use of regression analysis means that only the overall built-form trend can be characterised; *i.e.*, it is not possible to derive and hence analyse within a region-based representation of land-cover all of the built-form levels proposed by Krüger. This is mainly due to the fact that the approach adopted is

limited to a hierarchy that describes the spatial organisation of a constellation only in terms of the built-form units that fall within it (Barr and Barnsley, 2004).

According to Barr and Barnsley (2004), deriving regions from the initial classified raster map (1:2 500 scale topographic raster map; $1km$ cell resolution) resulted in different planimetric land-cover regions each of which consisting of multiple topologically connected dwellings. Given this fact, the problem is that it is not possible to characterise the topological interconnections that exist between the built-form units that comprise the built-form arrays of a constellation. In spite of this limitation, Barr and Barnsley (2004) emphasised that this form of representation does allow to a certain extent an analysis of built-form morphology, in terms of size and shape, which may potentially provide important discriminatory structural information to recognise different urban land-use types.

3.6 Summary

In this chapter, the basic principles of graph theory and its terminology were summarised. The main types of graphs, common families, and particular graphs to model specific situations relevant to this thesis were revisited.

The two graph-search algorithms available in the literature, used in the process of learning structural properties of graphs, were reviewed: depth-first search and breadth-first search.

In order to give an example of how graph theory can be used in spatial analysis, five different graph-based approaches were reviewed, namely: the *eXtended Relational Attributed Graph (XRAG)*, used for the inference of land-use information from a land-cover map; space syntax, which attempts to explain human behaviour and social activities from a spatial configuration point of view; *β-skeletons*, for the description of the internal shape of point patterns; two examples of multiobjective approaches for urban network analysis; and built-form connectivity, which attempts to explain the spatial organisation of an urban system on the basis of its distribution of buildings.

Previous Chapter 2 and present chapter give the background of the research described in this thesis. Next chapter states more formally the purpose of the research as well as its aims and objectives.

4 Thesis objectives and research process

In this chapter the need for additional research is expressed, and the purpose of the study is stated more formally. In order to answer the research questions identified, the aims and objectives of this thesis are summarised. Finally, an overview of the research process proposed concludes this chapter.

4.1 The need for more investigation

Research has revealed the importance of the concepts from the mathematical areas of topology and graph theory for interpreting the spatial arrangement of spatial entities. Graph theory in particular has been used in different applications of a wide range of fields for that purpose, however not many graph-theoretic approaches to analyse entities within the urban environment are available in the literature.

Very little work has been devoted to the interpretation of initially unstructured geospatial datasets. Indeed, in most of the applications developed so far for the interpretation and analysis of spatial phenomena within the urban context, the starting point is to some extent a meaningful dataset in terms of the scene. Starting at a level further back, before meaningful datasets are obtained, the interpretation and analysis of spatial phenomena, especially in the urban environment, are more challenging tasks and require further investigation.

LiDAR data were used in this research as one of the example scenarios to test the developed algorithm. LiDAR data are usually combined with other sources of data

for geographic information production. Although our research was not carried out from the perspective of the Remote Sensing field, and hence LiDAR analysis was not its main interest, further investigation still appears pertinent on the processing of LiDAR data as a single data source for the retrieval of higher-level geographic information.

The lack or even the non-existence of graph analysis functionalities in common GIS packages suggests the development of an interactive tool for the visual representation and analysis of such a spatial data structure.

Moreover, structured inquiries and spatial analysis are certainly integral to GIS, but they fail to emphasize the power of the human eye in detecting patterns and the role of subjectivity in GIS. In fact, in most of the automated and semi-automated methods the user is not able to visualise any interim results until the end of the whole process. It is believed that the capability of visualising the results of interim steps is relevant and constitutes an extra value.

More formally, the purpose of the research reported in this thesis was:

- To investigate whether it is possible to work with initially unstructured geospatial data, and hence no prior knowledge of the spatial entities is assumed in this case, in order to produce higher-level information.
- Although geometric information is inevitably implicit in the whole process, the development of a purely topological approach was sought. The question is: how far it is possible to go just by looking at topological relationships between the spatial objects?

4.2 Thesis aims

In the light of the research questions mentioned above, the aims of the thesis have been identified as follows:

- The consideration of the problem as a general task of finding higher-level structures in an arbitrary collection of lower level details.
- The retrieval of structured information translated into more meaningful homogeneous regions, which can be achieved by:
    - identifying meaningful structures within the initial random collection of objects;

  - and by understanding their spatial arrangement in terms of the topological relationships between polygons of adjacency, containment and touching.
- The investigation of the topological relationships between objects in the context of the whole spatial scene rather than within their individual neighbourhood (defined in this thesis as "higher-order analysis").
- In order to meet the aims stated above, a higher-order topological analysis method should be designed and implemented; this should be done primarily based on: the investigation of the spatial relationships beyond the first level of adjacency, and on the extension of the standard notion of adjacency to that of containment.
- The development of the algorithm needs to be as much as possible independent of the test environment.
- The extension of the range of the methods used in order to visualise the resulting graphs of adjacencies and present the urban scene topology. In this respect, the objective is to link the higher-order topological analysis method back to the original environment, *i.e.* the map of the spatial objects. For this, an interactive tool should be implemented in a GIS.

4.3 Research methodology

An overview of the research methodology required to answer the stated research questions and to meet the identified aims of the thesis, is presented in Figure 4.1 below.

The first step refers to the preparation of the raw data. In order to start structuring information and make it more explicit, some topological information was brought in to the original datasets by describing the internal shape of the point sets. This was accomplished by establishing a triangulated irregular network (TIN) through the given datasets, based on the Delaunay Triangulation algorithm. The TIN facets were then classified according to their gradient, and a map of gradient regions was generated for each dataset used.

The second step entailed the construction of the network of connectivity throughout the generated maps of gradient regions by applying graph theory. The interpretation of the spatial topology based on the extension of the standard notion of the relation of adjacency to that of containment between those regions, and also the consideration of

the touching relationship between "steep" polygons, led to the design of the *containment-first search* method for higher-order topological analysis.

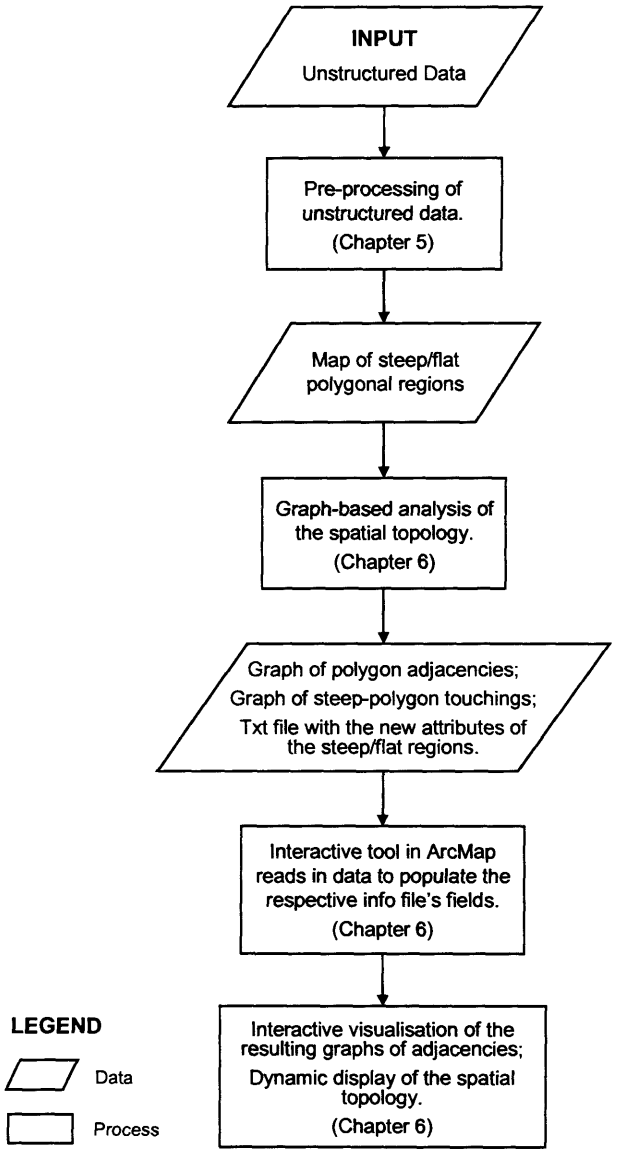


Figure 4.1– An overview of the research methodology.

Finally, the third step is related to the translation of the results of the previous step into something that can be visualised in the GIS environment. An interactive tool was developed in ArcMap for the spatial topological representation of initially unstructured geospatial datasets.

4.4 Summary

After establishing in Chapter 1 the overall context of the research reported in this thesis, and summarising in Chapters 2 and 3 the relevant literature reviewed, in this chapter the main issues that need further study were pointed out. The research questions were stated, and in order to answer them the main aims of the thesis were identified.

To conclude this chapter, an overview of the research process was given. The whole research process is described in detail in the next two chapters: Chapter 5 on the test data and the steps concerning their preparation; and Chapter 6 on the design of the higher-order topological analysis algorithm, and on the technical aspects of its implementation.

5 Test data

In most of the applications available in the literature, the input data are usually structured and attributed, and considered meaningful in terms of the geographical scene. In this work, one of the purposes is to explore and investigate the possibility of having input data that are unstructured and unattributed. The assumption here is that there is no prior knowledge of the spatial entities nor of their spatial arrangement.

In order to develop the algorithm for higher-order topological analysis, a test framework was needed. As an example scenario, *LiDAR* (*Li*ght *D*etection *a*nd *R*anging) data were mainly used. LiDAR is a type of unstructured data which does not generally provide imagery of the surveyed area[3]. Range data are a random collection of a considerable number of 3D points, with no pattern pre-defined, which can be used, for instance, for the automated generation of digital terrain models (DTMs). It should be stressed however that the method proposed in this thesis was not specifically designed for LiDAR analysis.

[3] Present laser scanning systems can provide image information taken by video cameras during the flight mission, which is mainly used for documentation and visual inspections. However, the quality and resolution of video is limited; also, a true spatial and temporal co-registration between laser and images is in most cases not accomplished, because video images are usually not integrated with the laser system (Ackermann, 1999; Baltsavias 1999b). As far as intensity data are concerned, the minor advantage is that, being produced by active systems, they are insensitive to illumination shadows; furthermore, laser intensity images are already geocoded, thus no orthoimage generation is needed. Nevertheless, given their lower object resolution, laser images cannot compete with high quality optical imagery (Baltsavias 1999a).

In section 5.1 below, the principles of LiDAR systems, the data and its main applications are briefly reviewed. The LiDAR datasets used for development and test purposes are described in section 5.2. In order to analyse the versatility of the developed algorithm, a few experiments were also carried out with photogrammetric data; the datasets used are presented in section 5.3. In section 5.4, the steps undertaken in the preparation of the raw data are explained and summarised both in the form of flowcharts and pseudo code. Finally, a summary of this chapter is given in section 5.5.

5.1 LiDAR system, data and applications

The development of LiDAR systems, or *Airborne Laser Scanning* (ALS) systems, goes back to the 1970s. Currently, is an established method with high technical and economic performance for the highly automated generation of DTMs (Ackermann, 1999).

In the 1980s, with the development of the *Global Positioning System* (GPS) positioning and the improvement of the ability to measure position and attitudes of *Inertial Measurement Units* (IMU), the study and application of this mapping technology have developed further, and have been spreading quickly into various practical applications beyond the DTM generation (Ackermann, 1999; Zeng *et al.*, 2002). In fact, ALS has had a fast and most successful development, mainly technology driven. According to Baltsavias (1999b), in 1996 there was only one company selling commercial ALS systems, and the service providers could be counted on the fingers of one hand. After three years, that number jumped to about 40 similar airborne laser scanning mapping systems in the world.

5.1.1 General principles

Like many multispectral scanners, laser scanning systems utilise opto-mechanical scanning assemblies; they constitute active sensing systems which use a laser beam as the sensing carrier (Wher and Lohr, 1999). These systems generate geometric results in terms of distance, position, attitude, and coordinates (Ackermann, 1999).

The measurement principal used in LiDAR is that of range measurement of an oriented laser beam generated from a platform whose three dimensional (3D) coordinates (*X, Y, Z*) are known; for each shot, the spatial vector from the laser platform to the illuminated spot on the ground is established, and the (*x, y, z*) coordinates of footprint are computed (Ackermann, 1999; Wher and Lohr, 1999).

LiDAR generates altimetric data by combining and integrating state-of-the-art sensors (*vd.* Figure 5.1): GPS receivers, an *Inertial Navigation System* (INS) and a powerful *Laser Range Finder* (LRF) (Eurosense, 2003). The scanner deflects the laser beam at right angles to the flight line resulting in a swath of ground being sampled. The position of the sensor is calculated from GPS and INS data. The 3D position of each laser beam spot on the earth's surface is determined in combination with the scan angle measurements (Baltsavias, 1999b).

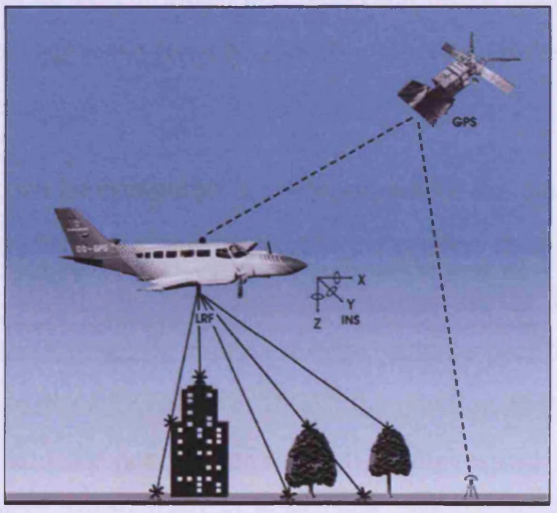


Figure 5.1 - Principle of LiDAR, or Airborne Laser Scanning.
(From Eurosense, 2003).

The integration of these systems mentioned makes the airborne laser altimetry mapping system a tool for high-speed and high-accuracy altimetric mapping with a short turn-around time.

In the middle of the 1990s, Professor Li Shukai (Chinese Academy of Science) presented the idea to integrate GPS, INS and laser scanner together with the spectral imaging scanner to form what was called *Airborne 3D Imager*. The distinctive advantage of Airborne 3D Imager is that it can produce geo-referenced spectral image and DEM data without any ground control points (Zeng *et al.*, 2002).

For airborne laser ranging there are two different systems in use, the *continuous wave laser* and the *pulse laser*, with the latter in use in most cases (Baltsavias, 1999c; Wehr and Lohr, 1999). Both systems are based on a distance measurement between the sensor system and the ground, and take into account the travelling time of a signal; however, different physical effects are utilised. These measurements take place within short intervals and yield many single distances that are incidental to a certain footprint on the ground. The pulse laser system sends out single pulses, which are reflected on the ground; the continuous wave laser system is based on the measurement of the phase difference between a continuous outgoing and incoming wave train (Baltsavias, 1999c; Wehr and Lohr, 1999).

After the survey flight, there are two different datasets available: the positions from the GPS/IMU and the laser ranges with the dedicated instantaneous scanning angles. These datasets have to be synchronised and then each range has to be combined with its respective position achieved by the GPS/IMU system (Baltsavias, 1999c; Wehr and Lohr, 1999).

But, before any data can be computed, all components of the measurement unit must be calibrated. For the purpose, some systematic parameters must be considered, such as (Wehr and Lohr, 1999): the three mounting angles of the laser scanner frame, described by the Euler angles roll, pitch and yaw, with respect to the platform-fixed coordinate system (usually with origin at IMU); the position of the laser scanner with respect to the IMU; and the position of the IMU with respect to the GPS. This so called "calibration data" can be derived from laser scanner surveys, whereby certain reference areas are flown over in different directions (Wehr and Lohr, 1999). From the relative orientation and position of the different surveys, and their absolute orientation and position with respect to an earth-fixed coordinate system, calibration data can be derived (Wehr and Lohr, 1999). According to Wehr and Lohr (1999), there is no standard procedure for calibration though, and each laser scanning firm has its own. Nevertheless, Lindenberger (1993), (cited in Wehr and Lohr, 1999), described typical calibration procedures.

According to Lindenberger (1993) and Hug (1996), (both cited in Wehr and Lohr, 1999), and Baltsavias (1999c), single ground points in the WGS84 reference system can be calculated by using three datasets: calibration data and mounting parameters;

laser distance measurements, with their respective scanning angles; and the position of the laser platform.

The survey results in a cloud of randomly distributed laser points, in plan and height, covering the flown over area (Ackermann, 1999; Axelsson, 1999; Baltsavias, 1999c; Wehr and Lohr, 1999). In fact, laser scanning is not capable of any direct pointing to particular objects or object features; the resulting coordinates refer to the footprints of the laser scan as they happen, and only visible ground surface or objects on it are directly measured. Elevation data can be acquired with different attributes depending on application and laser scanner system. Some of these attributes are (Axelsson, 1999):

- Point density;
- Registrations of multiple echoes;
- Amplitude registration (reflectance).

The point sampling density of LiDAR systems and distribution pattern of the points depend on several factors. These include the height and velocity of the platform the laser scanning system, and field-of-view and sampling frequency of the sensor (Ackermann, 1999; Axelsson, 1999; Baltsavias, 1999c; Steel *et al.*, 2003). According to the Environment Agency (1997, 1998; cited in Steel *et al.*, 2003), sampling intervals of <3m on the ground and a vertical accuracy of 10cm-15cm are typical, however greater levels of detail and accuracy can be achieved depending on the factors mentioned above.

Multiple echoes of one laser pulse can be registered by some systems. This can be of importance in filtering and modelling algorithms related to "solid" objects (*e.g.* buildings), "light-permeable" (*e.g.* vegetation canopies) and ground surface separation (Axelsson, 1999; Steel *et al.*, 2003). Multiple echoes can also be found at the edge of buildings, thus indicating a very fast change in elevation (Axelsson, 1999).

The last attribute, amplitude or reflectance registration, gives radiometric information about the surveyed area. Depending on the spectral reflectance properties and the wavelength of the laser, amplitude information can, in principle, assist in distinguishing different surface cover types, *e.g.* paved areas from grass land (Axelsson, 1999; Steel *et al.*, 2003).

Baltsavias (1999b) noted that the specifications of some parameters concerning accuracy made available by vendors are often unclear. Some firms give only range accuracy or simply accuracy, without separate indication for planimetric and height accuracy. Nevertheless, Ackermann (1999) argued that the overall vertical system accuracy is usually in the dm order. In addition, Wehr and Lohr (1999) pointed out that as laser scanners have a potential range accuracy of better than 1 dm, the position system should allow at least the same accuracy. Moreover, it is known that the horizontal accuracy is better than 1000^{th} of the flying height (USACE, 2002), and considering that most systems presently operate at flying heights up to about 1000m above the ground (Ackermann, 1999), it can be said that the planimetric accuracy is at least better than 1m.

Concerning ALS, Baltsavias (1999c), Wehr and Lohr (1999) have described basic relations and formulas as well as parameter calculation methods, including point density (which is considered the key factor of ALS data processing).

5.1.2 Applications of LiDAR data

DTMs are one of the most typical and immediate applications of range data, which in turn are used for a wide range of purposes (Ackermann, 1999; Baltsavias, 1999a; Zeng *et al.*, 2002; Steel *et al.*, 2003). The generation of DTMs using range data has clear advantages over more classical methods, such as photogrammetry (Ackermann, 1999; Baltsavias, 1999a; Eurosense, 2003):

- It is independent of light conditions and less dependent on weather conditions compared to, for instance, aerial photography, because it is an active measurement system; LiDAR or laser scanning flights can be executed during the night;
- It is equally applicable to open terrain as well as to areas which are partly or completely covered by dense vegetation; in fact, the terrain can still be mapped even when it cannot be seen on aerial images, because the ALS system also has a certain capacity to penetrate vegetation;
- Other advantages are the sparse field work, apart from putting up a GPS receiver in or near the project area, and the short data processing time because of reduced operator interaction, allowing a shorter turn-around time.

Ackermann (1999) also noted how laser scanning is important in generating DTMs in coastal areas or wetlands which are difficult to be obtained by other methods.

As far as urban applications are concerned, LiDAR systems have been employed so far in various studies of urban areas, typically to generate models of building elevation or to segment the scene into discrete land-cover classes by man-made (building) or other landscape (tree) object detection. The accuracy with which these can be achieved is dependent on the relationship between the LiDAR point-sampling density and the size (scale) of the objects to be classified or modelled (Steel *et al.*, 2003).

A particularly interesting and relatively recent application of ALS concerns the automatic capture of buildings in built-up areas for city modelling purposes (Ackermann, 1999; Wehr and Lohr, 1999). According to Ackermann, man-made structures, masking the ground surface, were originally considered as obstructions to be removed in the DTM generation. In the meantime, the detection and capture of buildings and vegetation has become an important independent task (Axelsson 1999; Nardinocchi *et al.*, 2003; Forlani *et al.*, 2006).

Currently, building detection methods are being enhanced with the data fusion between range data and optical images. According to Kim and Muller (2002), besides other datasets such as aerial photographs, GIS information and pre-existing DEMs, LiDAR data can also be used to complement, for example, IKONOS images (*vd.* also Sohn, 2004). Actually, in comparison with previous satellite image sources (like SPOT and Landsat), new high resolution satellite images, such as those from IKONOS, provide potentially useful information for the identification of individual surface objects, such as trees and buildings. In particular, it may be possible to utilise LiDAR scanning altimetry data as an alternative source of 3D information to offset the current paucity of IKONOS stereo-pairs. Range data offer the added advantage of providing information on building height and, in some instances, roof-form (*i.e.* pitched or flat).

The promising approach of integrating multi-spectral satellite image data with range data has also been tried by some authors, including Zeng *et al.* (2002) and Steel *et al.* (2003), in an attempt to solve the challenging task of urban land-use classification.

Zeng *et al.* (2002) applied height data acquired by airborne laser scanning for geometric correction of the multi-spectral satellite image by the generation of ortho-images; they also integrated it with urban environment classification. In their work, range data acquired by Airborne 3D Imager was used to obtain information on the location height above the terrain surface for each pixel. Afterwards, this information was applied in order to separate urban objects higher than the ground level from objects that are at ground level, *e.g.* buildings, trees, streets, grass-covered areas, water bodies and bare land, etc.

In Steel *et al.*'s (2003) work a combination of range data and satellite multi-spectral scanner data is used to derive information on buildings present in two residential and two industrial districts of Cardiff (Wales, UK). Then XRAG (introduced in section 3.5.1) is employed to derive from these data information on the structural composition of land-use. This information is used to analyse the extent to which these two urban land-use categories can be identified and distinguished on the basis of their structural composition.

The high level of detail and accuracy that characterise LiDAR data and its cost-effectiveness, specially for large areas when compared to the cost of conducting a similar survey using ground-based techniques (Ackermann 1999; Wehr and Lohr, 1999; Steel *et al.*, 2003), have encouraged the use of range data in a diverse set of applications ranging from studies of vegetation canopy structure (Blair *et al.*, 1999, cited in Steel *et al.*, 2003), surface hydrology (Ritchie, 1996, cited in Steel *et al.*, 2003) and ice sheet dynamics (Krabill *et al.*, 1995, cited in Steel *et al.*, 2003) through to detailed topographic survey (Krabill *et al.*, 1994, cited in Steel *et al.*, 2003).

5.2 Description of the LiDAR datasets

5.2.1 The London dataset

For the purposes of conception and implementation of the graph-based tool, a range dataset[4] representing an area in Kew, southwest London, including the National Archives building, was mainly used. It is a 3D dataset that contains filtered laser returns. The data have been thinned out by the provider to give a point spacing of about 3m, and it contains both ground points and object points reflected from trees, buildings and other small objects above ground level, like cars. The whole area ($1470 \times 1530 m^2$) comprises a total of 169819 laser points, which results in an average point density of approximately 1 point/$12m^2$.

Once obtained, loaded and processed, LiDAR data can be viewed in different ways. One common way is to view it in its native point (x, y, z) format, which allows the user to understand the point density and spatial distribution of the data. Colour coding the image according to the points' height also gives one an idea of the height range of the data. Figure 5.2 depicts the cloud of points colour coded in accordance with the points' height. It can also be seen in the same figure that the point separation of 3m mentioned above is not homogeneous, for there are areas where no returns were registered (like on the west side of the National Archives building, corresponding to a pond, and across the River Thames)[5].

[4] Provided by the National Remote Sensing Centre, Southwood, UK; made available by the Department of Geomatic Engineering of the University College London for academic purposes.

[5] Since infra-red beams are absorbed by the water, water bodies do not reflect LiDAR beams unless hydrographic LiDAR is used (Irish and Lillycrop, 1999).

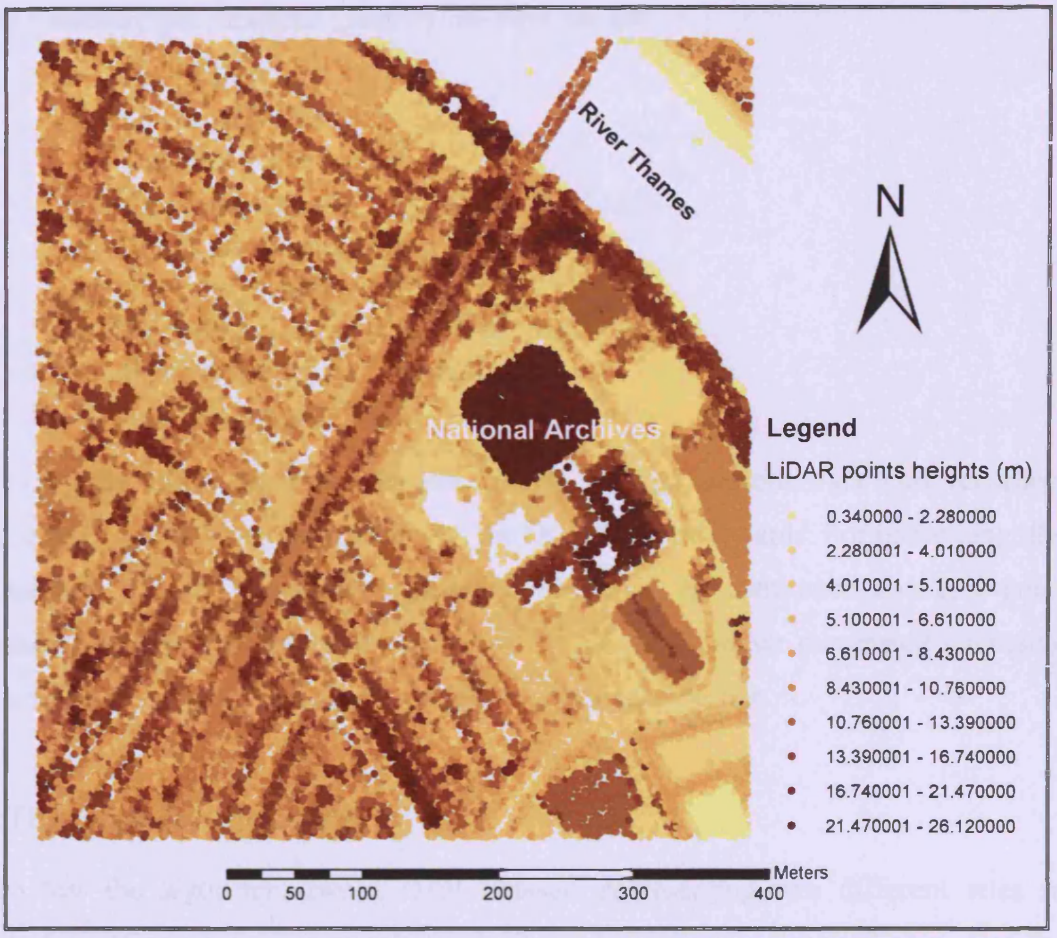


Figure 5.2 - LiDAR dataset used for development purposes of the graph-based technique – Kew, southwest London. Data is colour coded in elevation range.
(From de Almeida et al.*, in press)*

The dataset was acquired using a Navajo Chieftain fixed wing aircraft at a height above terrain of approximately 800m, using an Optech ALTM 1020 (Airborne Laser Terrain Mappers) laser scanner. Combining the flight height mentioned above (h) with a scan angle (θ) of ±20°, results in a swath width (SW) of 580m (*vd.* Equation 5.1; Baltsavias, 1999c):

$$SW = 2h \tan\theta . \qquad \textit{(Equation 5.1)}$$

Swaths are flown with a 30% overlap.

Taking into account the same flight height (h) and assuming a beam divergence of $0.3\text{x}10^{-3}$rad (γ), the diameter of the laser footprints (A_L) is about 25cm (*vd.* Equation 5.2; Baltsavias, 1999c):

$$A_L = h\gamma \qquad \textit{(Equation 5.2)}$$

The scanner specifications given by the provider are:

- PRF = 5 kHz;
- Scan angle = ±20°;
- Scan frequency = 12 Hz;
- Elevation accuracy (σ_z) better than 15cm;
- Range resolution = 1cm;
- Angle resolution = 0.1°;
- Horizontal accuracy (σ_{xy}) better than 0.8m;
- Beam divergence = 0.3mrad;
- Eye-safe range = 330m.

During the survey, the raw laser data, DGPS and INS position data were recorded. The raw tape data was combined with the DGPS data afterwards. For processing, the Optech's ALTM 2.26 software package was used. As mentioned above, a point spacing of 3m was specified during the output stage, which eliminated excessive points in the overlap areas and kept files to a manageable size.

5.2.2 The Stuttgart datasets

To test the algorithm, two LiDAR datasets representing two different sites in Stuttgart (Germany) were mainly used. These datasets were acquired as part of the second phase of the OEEPE (now EuroSDR) project on laser scanning (OEEPE, 2000). Like the London dataset, the Stuttgart datasets contain laser returns from both ground points and object points reflected from trees, buildings and other small objects above ground level.

"Site 1" covers an area in the city centre which extends approximately 420m in the east-west direction and 633m in the north-south direction. The Site 1 area has buildings of different shapes, sizes and orientation; there is also a fair amount of vegetation of different heights and densities. The associate dataset comprises 243400 3D points with a point spacing of about 1m. Figure 5.3 depicts the cloud of points colour coded in accordance with the points' height.

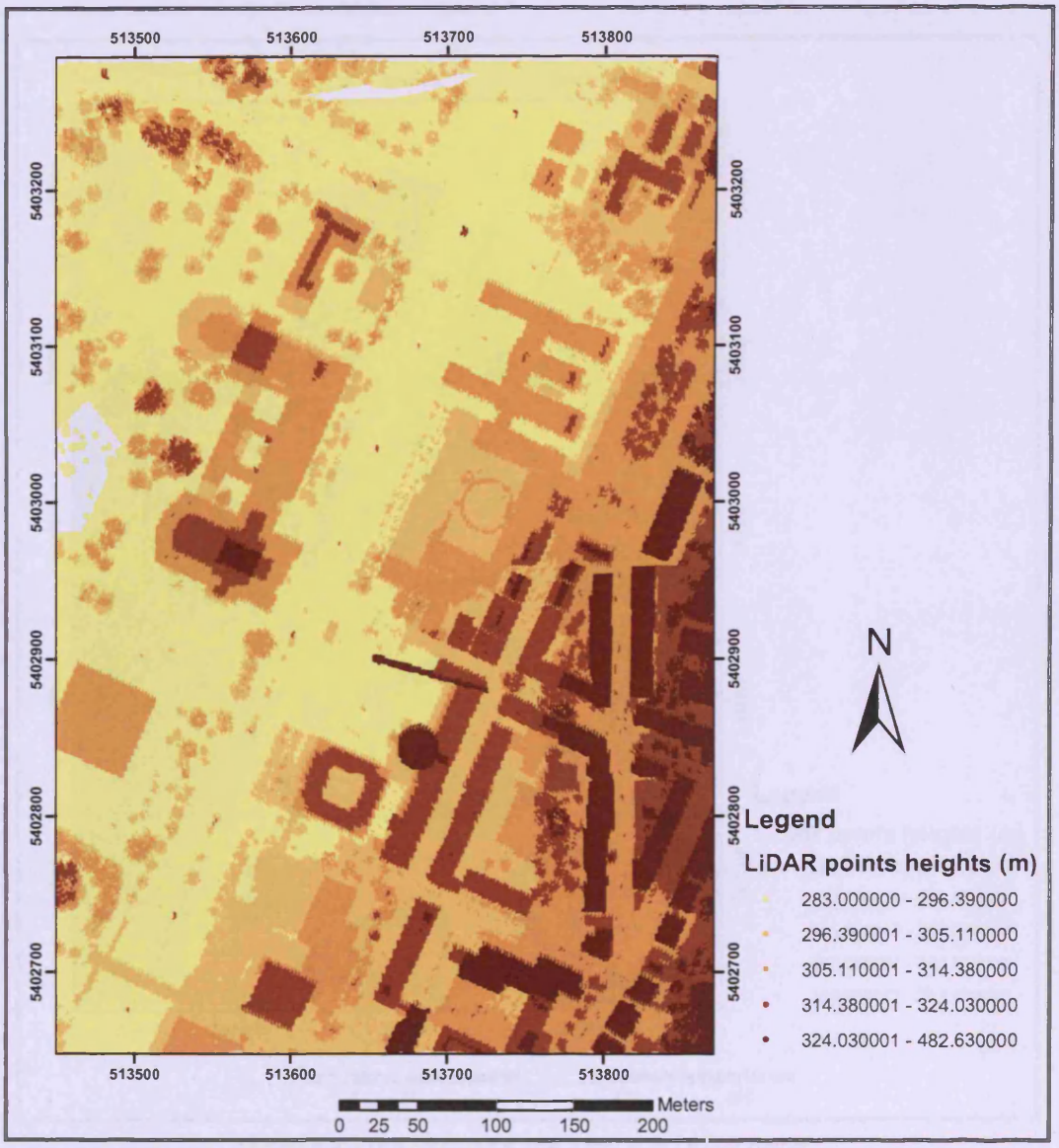


Figure 5.3 – The Stuttgart's Site 1 cloud of LiDAR points.

"Site 2" is mostly a residential area spanning approximately 650m in the east-west direction and 330m in the north-south direction. This site was chosen as a test area because, although it is still an urban site, there is a large presence of vegetation; buildings of different shapes, sizes and orientation are also present. The associate dataset comprises 188514 3D points also with a point spacing of about 1m. Figure 5.4 depicts the cloud of points colour coded in accordance with the points' height.

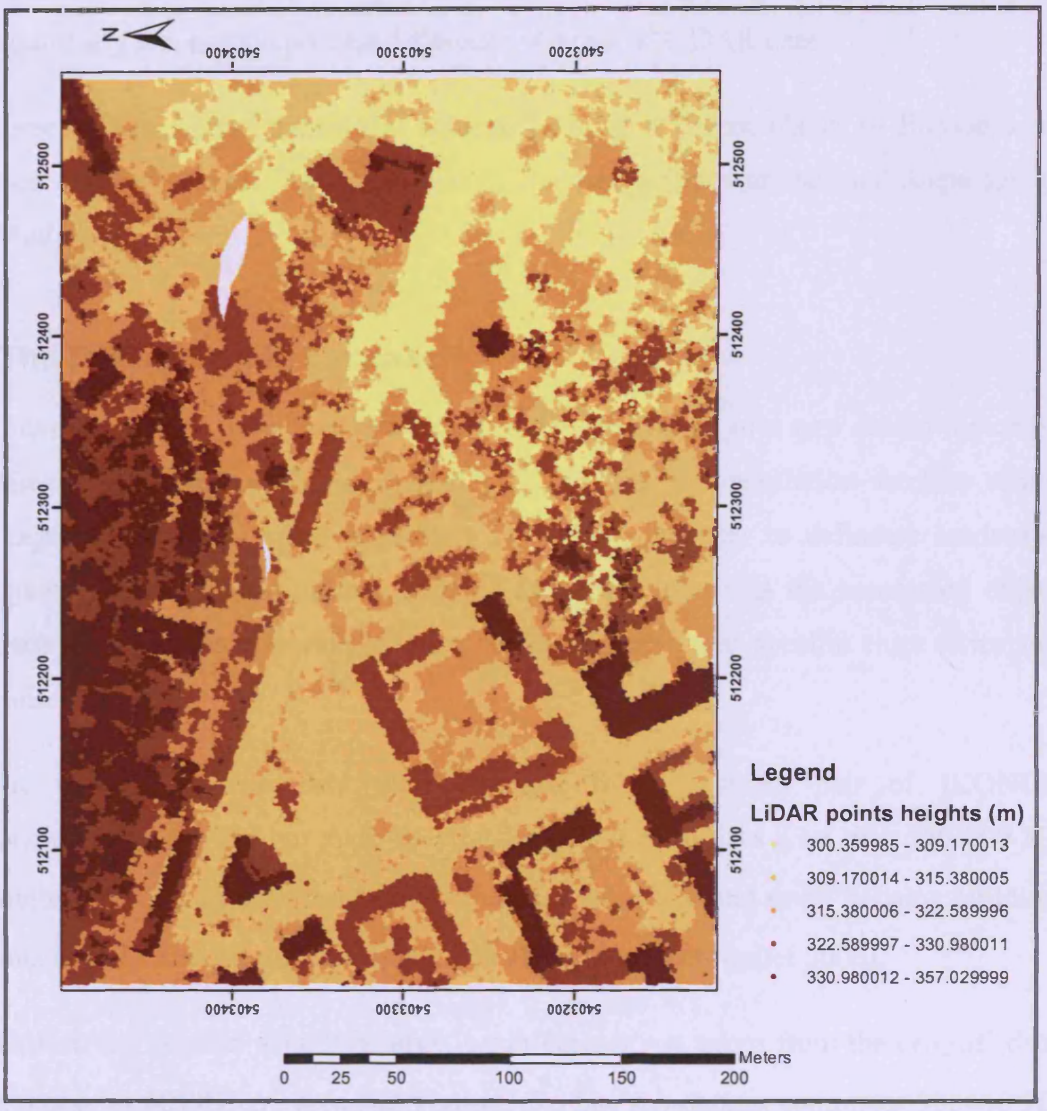


Figure 5.4 - The Stuttgart's Site 2 cloud of LiDAR points.

An Optech ALTM scanner was used in the survey, and both first and last pulse data were recorded (Sithole and Vosselman, 2003). Only first pulses of the OEEPE datasets were used for the current experiments. This was fundamentally due to the fact that first pulse LiDAR data have more information referring to vegetation than second pulse data.

5.3 Description of the photogrammetric datasets

In order to test the versatility of the algorithm developed in this thesis, other types of unstructured data were considered, such as photogrammetric data. Consisting of a cloud of 3D points, a photogrammetric dataset has the same structure as that of a

LiDAR dataset. Thus, for the purposes of this study, photogrammetric data did not require any preparation process different from that of LiDAR data.

Three different photogrammetric datasets of three different places in Europe were used: Barton-Bendish (Eastern England), Heerbrugg (Switzerland) and Santa Lucija (Malta).

5.3.1 The Barton-Bendish dataset

This dataset[6] was generated as a result of the application of a new stereo matching scheme, proposed by Kim and Muller (2006), for high-resolution satellite stereo image of residential areas (especially IKONOS), in order to delineate landscape object boundaries. The ultimate goal of this application was the automated object detection of trees and buildings when combined with multi-spectral clues (Kim and Muller 2006).

The photogrammetric data were obtained from a stereo pair of IKONOS panchromatic 1m and 4m multi-spectral images. It comprises a set of 1 560 000 3D points, covering an $11 \times 11 km^2$ area with dense woodland and small housing building units in Barton-Bendish, East Anglia - England (Kim and Muller 2006).

Because the original dataset is large, a sub-dataset was taken from the original data covering an 501 $815m^2$ area (*vd.* Figure 5.5). The sub-dataset comprises 506460 3D points which are colour coded in elevation range in Figure 5.6.

[6] An acknowledgement is due to Jung-Rack Kim (Department of Geomatic Engineering of the University College London) who made available the dataset for the purpose of this study.

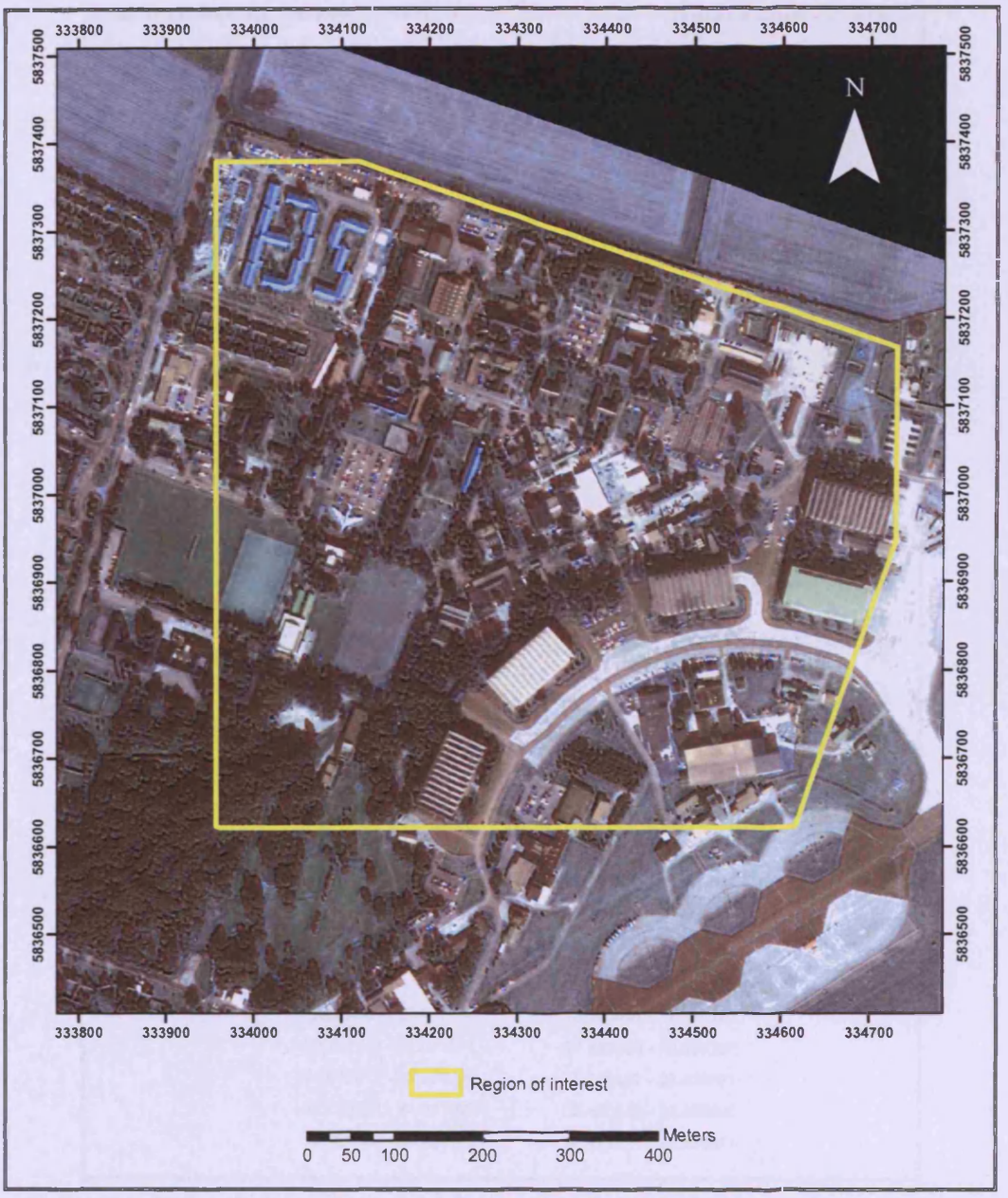


Figure 5.5 –The area of interest in Barton-Bendish.

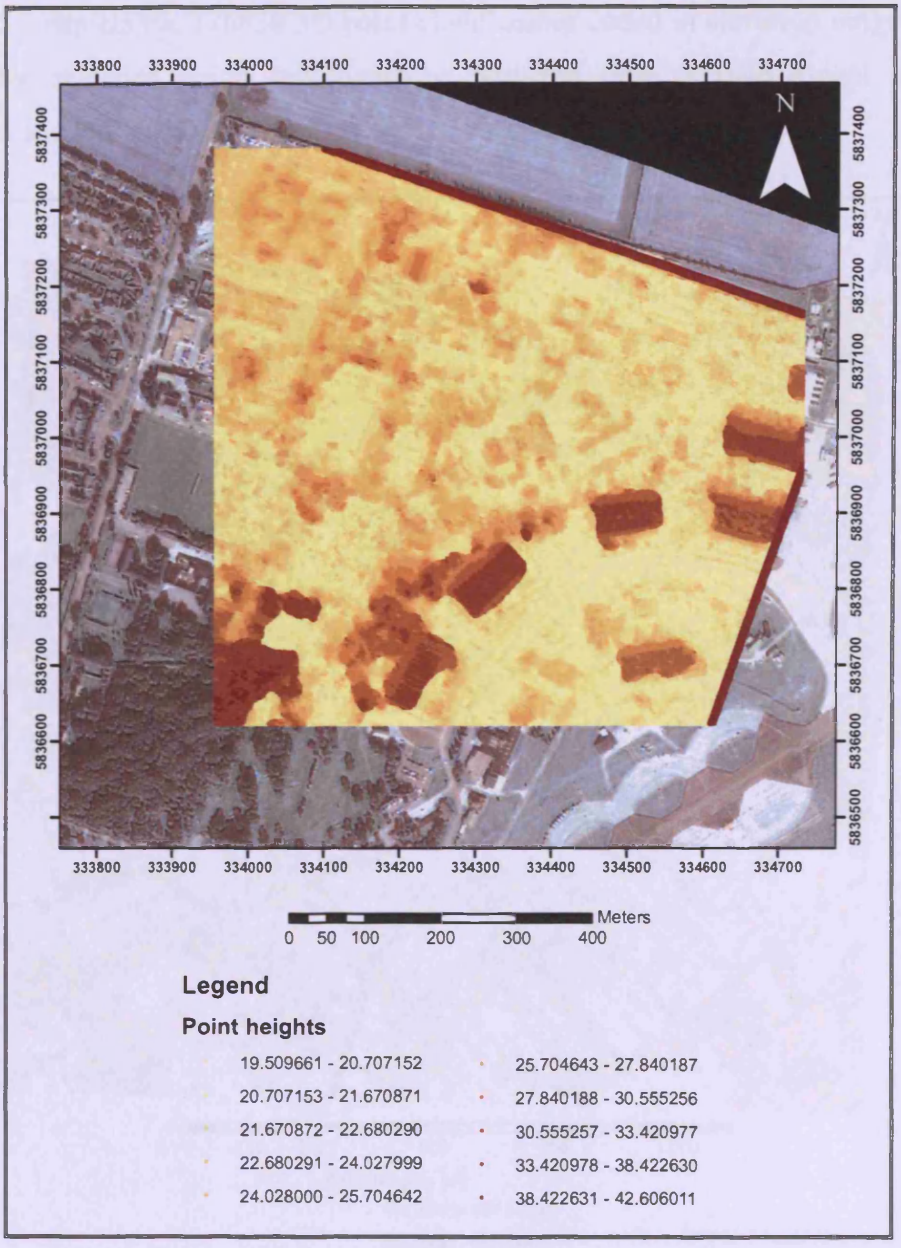


Figure 5.6 – The photogrammetric 3D point cloud of Barton-Bendish.

5.3.2 The Heerbrugg dataset

The surveyed area extends approximately over 1790m in the east-west direction and 1720m in the north-south direction.

Figure 5.7 depicts the 176836 3D point cloud colour coded in elevation range. These are stereo matched points automatically extracted from ADS40 digital imagery (Kokkas and Dowman, 2006)[7].

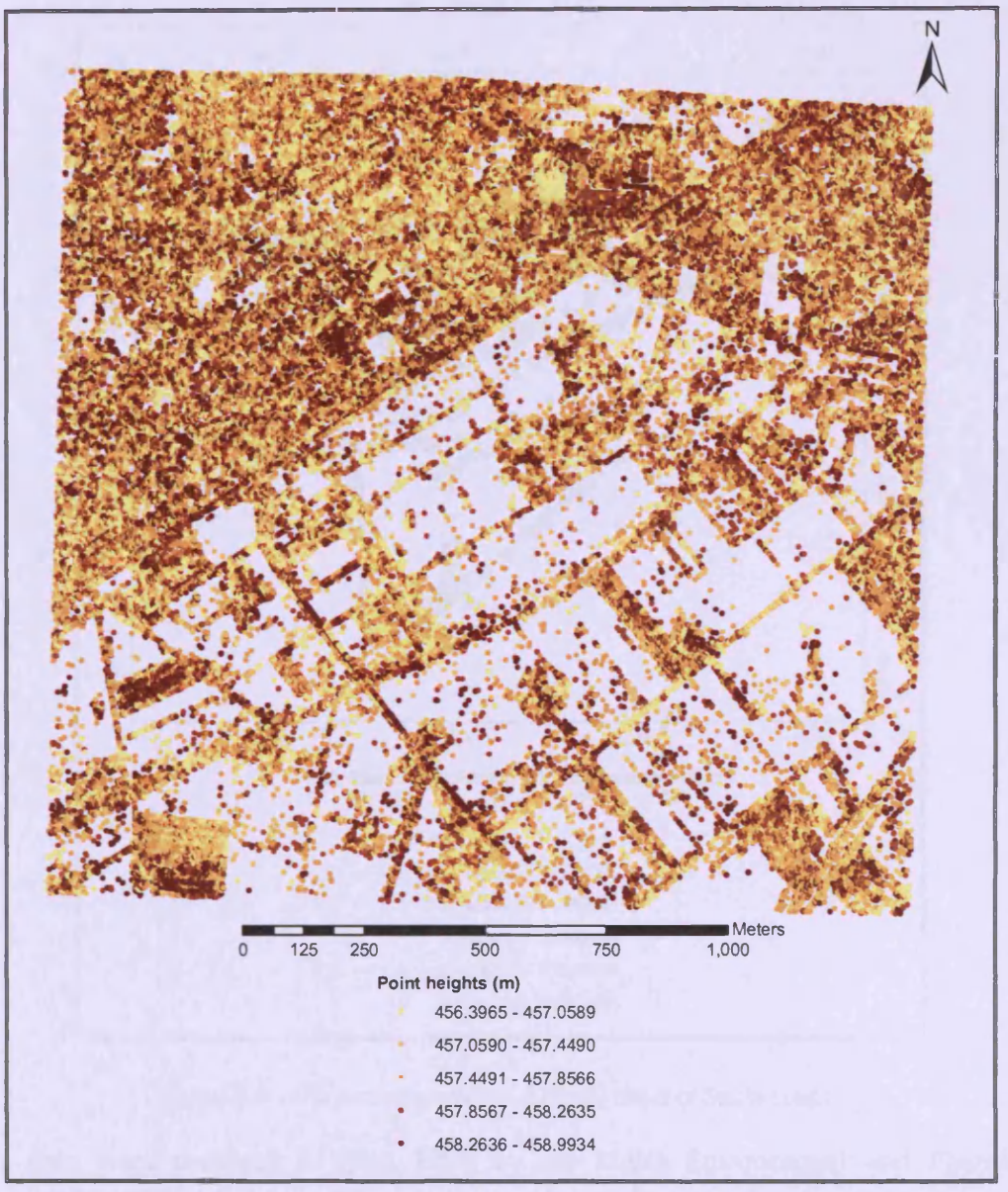


Figure 5.7 - The photogrammetric 3D point cloud of Heerbrugg.

[7] An acknowledgement is due to Nikolaos Kokkas (School of Civil Engineering of the University of Nottingham) who made available the dataset for the purpose of this study.

5.3.3 The Santa Lucija dataset

The surveyed area extends approximately over 230m in the east-west direction and 273m in the north-south direction.

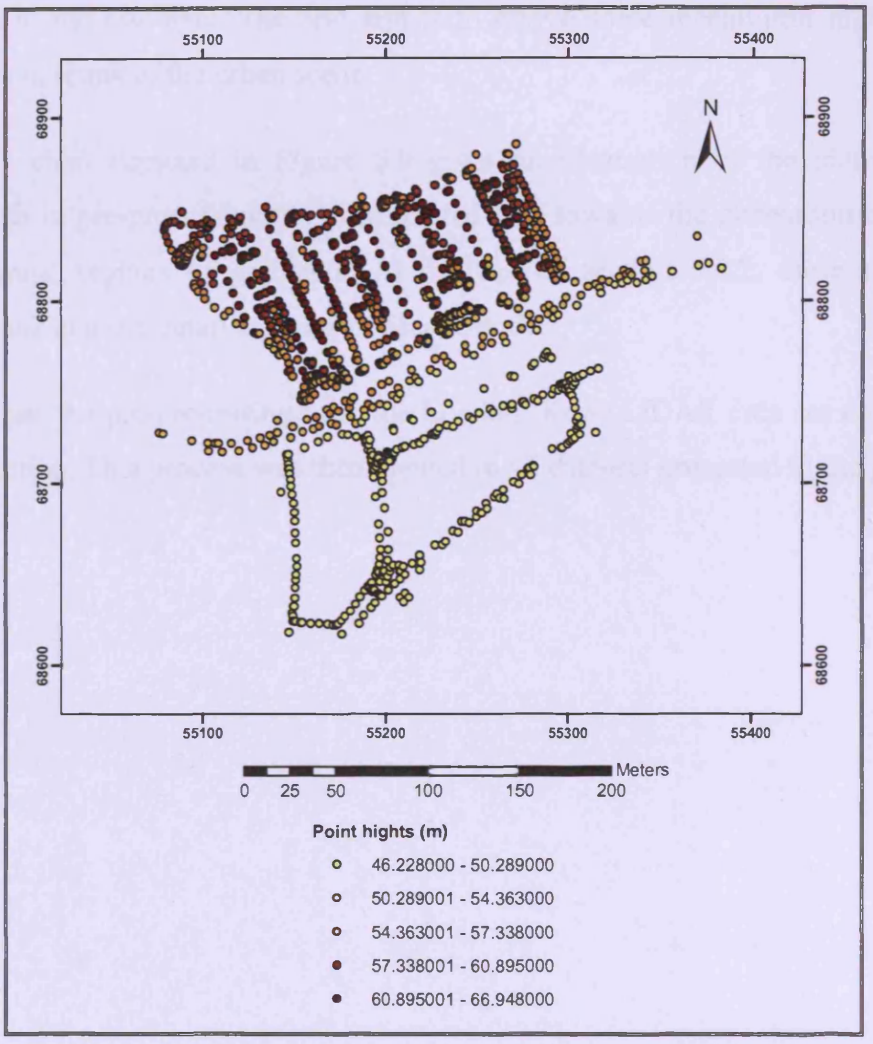


Figure 5.8 - The photogrammetric 3D point cloud of Santa Lucija.

The data were acquired in May 1994 by the Malta Environment and Planning Authority (MEPA)[8] for 1:1000 large scale topographic mapping: source, aerial photography - scale 1:4000; capture method, photogrammetric; stereoplotter, Wild AG1; software, Demeter PH. Figure 5.8 depicts the 1343 3D point cloud colour coded in accordance with the points' height.

[8] An acknowledgement is due to Carol Agius who made the data available by the permission of the Mapping Unit, MEPA, St Francis Ravelin, Floriana (Malta), www.mepa.org.mt. Data copyright MEPA and cannot be reused without the prior permission of the Mapping Unit.

5.4 Pre-processing

As can be seen in Chapter 4, the preparation of the data does not constitute one of the main aims of this study; however, is a fundamental starting step to give some structure to the raw data. The real aim is to derive some meaningful higher-level structures in terms of the urban scene.

The flow chart depicted in Figure 5.9 gives an illustration of the methodology undertaken in pre-processing the unstructured data towards the generation of a map of polygonal regions of gradients. At the end of section 5.4.2, these steps are summarised in more detail in terms of pseudo code.

To illustrate the pre-processing task, the London (Kew) LiDAR data are considered in this section. This process was then applied to all datasets presented in the previous sections.

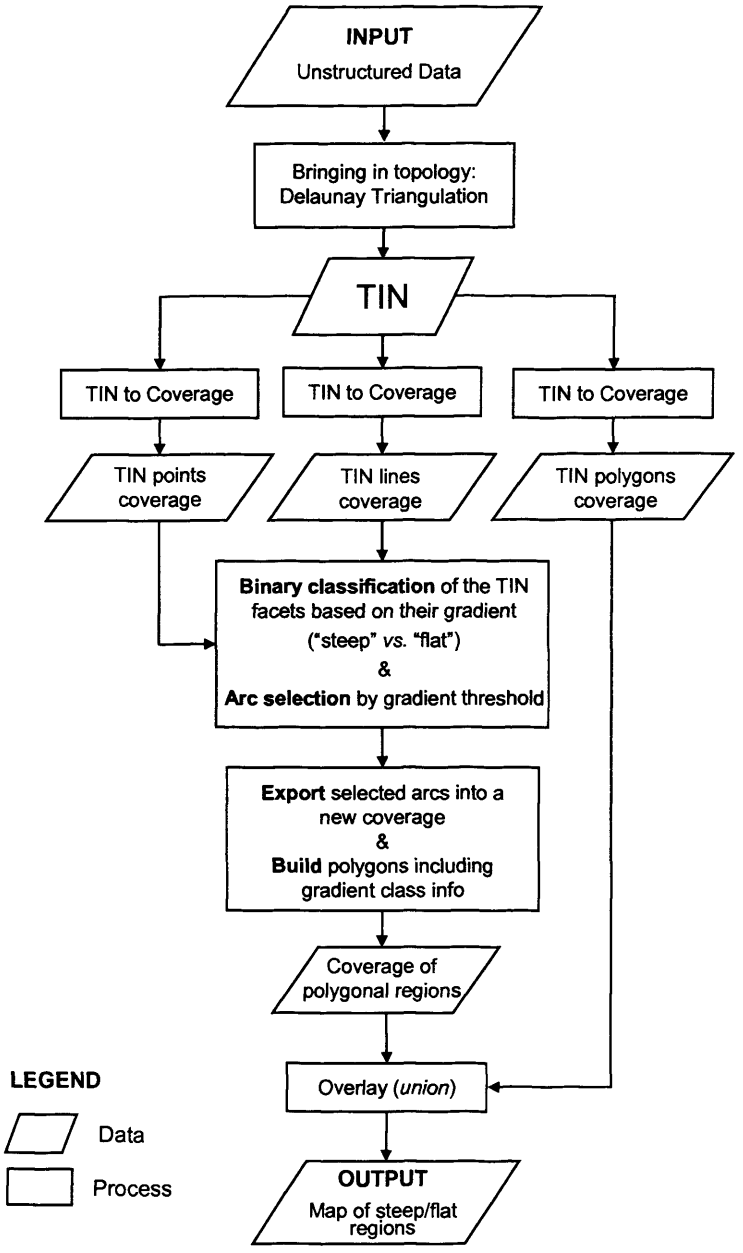


Figure 5.9 – Overview of the main steps of the pre-processing of unstructured data.

5.4.1 Internal description of the point patterns

In order to start structuring information and make it more explicit, some topological information was brought in to the original datasets by establishing a triangulated irregular network (TIN) through the given dataset (*vd.* associated DSM in Figure 5.10). The generation of this TIN was based upon the Delaunay triangulation (DT). The DT is a maximal planar description of the given point set's internal structure (Kirkpatrick and Radke, 1985), thus the TIN expresses proximities and neighbourhoods between the 3D points. This was accomplished with the 3D Analyst

extension of ArcMap (ArcGIS 8.3 environment), using as input an ArcInfo coverage containing each point set described above.

From the GIS analysis perspective, a TIN basically translates the original unstructured data into what is defined within this thesis as a set of *first order connections* in vector domain, *i.e.* a set of spatial relationships between objects in direct contact. In this research work, by using a graph-based approach, the plan was to build up networks of connectivity throughout these processed datasets to allow the performance of *higher-order connectivity* analysis, *i.e.* to investigate and understand the spatial arrangement between objects within the context of the whole geographical scene, rather than within the context of their individual neighbourhood (de Almeida *et al.*, in press).

Thus, the next step is to understand the meaning of the arrangement of the TIN facets by initially observing their adjacent neighbours, *i.e.* in direct contact, and by observing thereafter particular sequences of links between them. For instance, does a particular sequence of links between TIN triangles reveal the existence of any urban feature, or it is simply an open area? In other words, does that sequence mean an *off terrain* or *on terrain* area? In a further stage, though not covered in this thesis, it may be even possible to extend the analysis in order to characterise neighbourhoods and distinguish different urban functions, in other words land-use parcels.

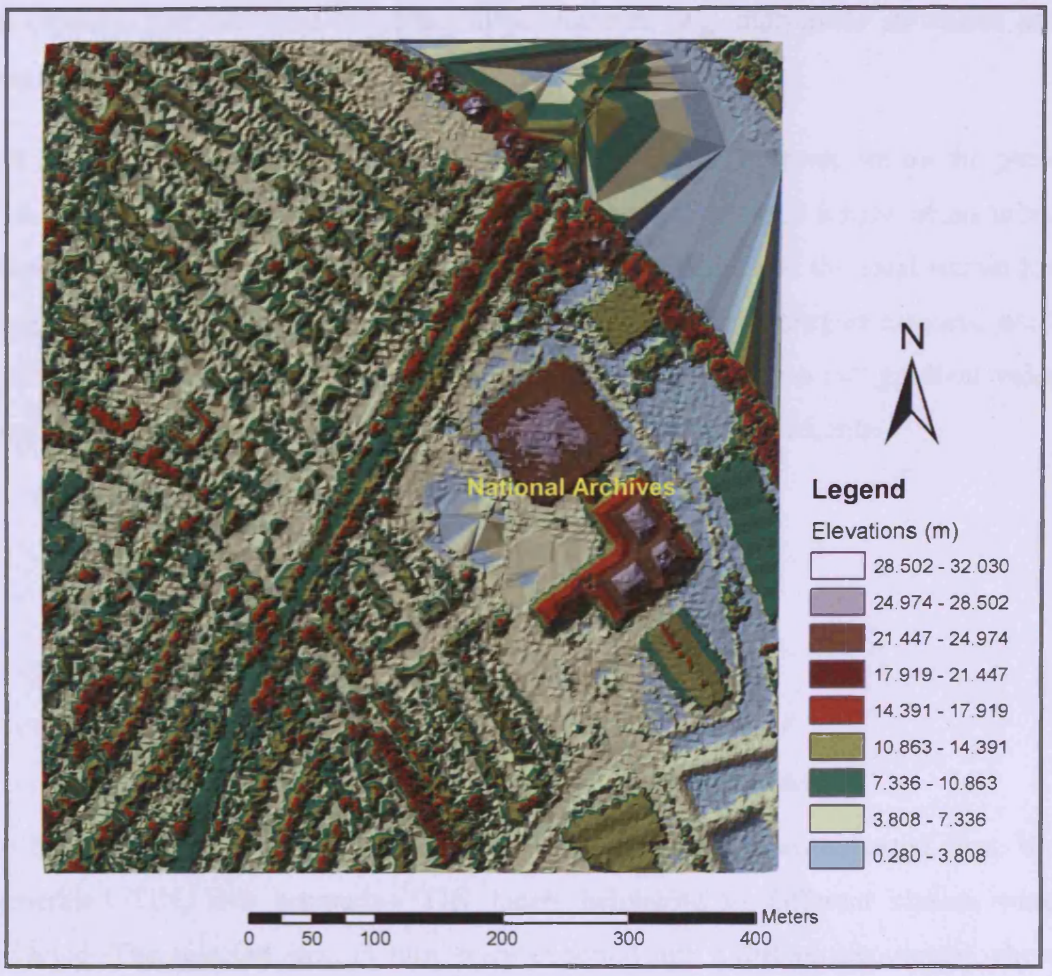


Figure 5.10 - DSM generated from the LiDAR point set of Kew - southwest London. (From de Almeida et al., *in press)*

5.4.2 Preparation of the polygon data base

To start with, the classification of the TIN facets and their aggregation into different polygons (to be mapped in one of the layers of the current ArcMap project) is proposed. This classification is based upon TIN facet attributes. Different criteria can be taken into account, for instance, their gradients, elevations, aspects or areas: it can be based on one or more of these geometric parameters; two or more classes with equal or different widths can be considered.

In this research, a binary classification of the TIN facets based upon their gradient was employed. As mentioned above, more than two classes could have been considered, but the problem is clearly simplified by considering only two gradient classes. In considering two classes of gradients, *i.e.* "flat" and "steep" TIN facets, it

is expected that the most important urban features (*e.g.* man-made structures and vegetation) are enclosed by the steep facets.

As the point spacing of, for instance, the London dataset is about 3m on the plane (1m for the Stuttgart datasets), and supposing that the average height of an urban feature is about 5m, a TIN facet straddling an urban feature and the local terrain has roughly a 60° gradient (80° gradient when considering the Stuttgart datasets, etc.), (*vd.* Figure 5.11). Thus, with regards to the London dataset, the 60° gradient value was initially considered for a binary thresholding of the facet gradients.

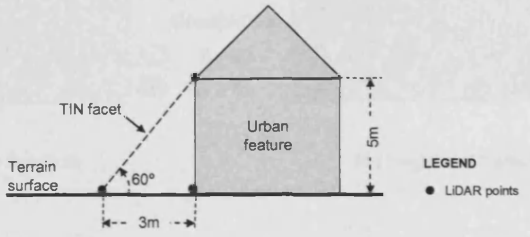
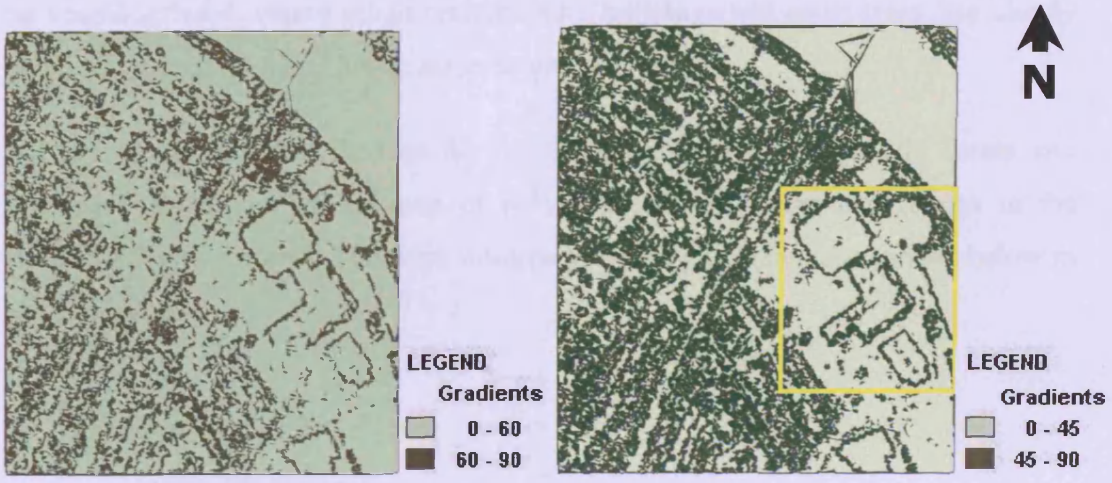


Figure 5.11 –The TIN facets binary classification thresholding.

In the ArcInfo coverage containing the TIN arcs, which were exported from the generated TIN, arcs separating TIN facets belonging to different classes were selected. The selected arcs, in turn, were exported into a different coverage where polygon topology was built, and hence several polygonal regions were created. In practical terms, it can be said that the polygonal regions were generated as though the TIN facets were aggregated in accordance with the binary classification detailed above: facets of the same class sharing an edge were "merged"; facets of the same class meeting at a node were preserved (de Almeida *et al.*, in press).

As can be seen in Figure 5.12a, building features are not well defined with this thresholding: this is especially evident in the eastern area containing the National Archives building and surrounding buildings. This fact was probably caused by the variation in facet gradient given the non-uniform distribution of the LiDAR points. Moreover, after having a look at the TIN facets gradient statistics graph, it was realized that the great majority of the facets have a gradient less than 30°. Therefore, a second experiment was carried out using a lower gradient threshold. The National Archives building and other major buildings were much better defined with a 30° gradient thresholding; however, the setback in using such a low value is that more noise is brought in. Given this fact, a 45° gradient threshold was considered and an

equal interval binary classification was performed. The result obtained with this new classification is shown in Figure 5.12b.



a) Using the 60º gradient threshold

b) Using the 45º gradient threshold

Figure 5.12 – Binary classification of TIN facets – Kew, southwest London.
(Box in yellow shows area extracted for analysis in Figure 6.3)
(From de Almeida et al.*, in press)*

Clearly, some of the urban features, like buildings standing on their own, are better shaped by polygons made of steep facets (45º-90º gradient class). Nevertheless, given the particular characteristics of the urban scene, the map of polygons obtained still constitutes a complex reality to be interpreted. For instance, residential areas, typically with a high density of small size urban features, cannot be recognised (*vd.* a detail of the southwest side of the surveyed area, Figure 5.13).

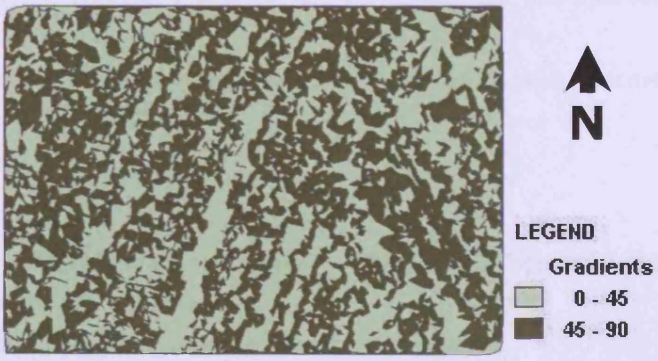


Figure 5.13 – A detail of the southwest side of the surveyed area depicted in Figure 5.12b).

Given the large size of the initial dataset, both in terms of area surveyed and number of LiDAR points, and the complexity of the map of polygonal regions, a case study

area was chosen for the purposes of developing the graph-based tool (*vd.* yellow box in Figure 5.12b). This case study area comprises the National Archives building and its neighbourhood, where urban features, like buildings and some trees, are clearly standing on their own and hence easier to analyse.

All the procedures described so far for the classification of the TIN facets and respective generation of the map of polygonal regions, were carried out in the ArcGIS 8.3 environment. The steps towards those purposes are summarised below in terms of pseudo code:

- Creation of the TIN.

  ArcMap (3D Analyst)

  `Create TIN from features` (Input: ArcInfo coverage containing the range point set).

- Creation of three ArcInfo coverages from the TIN containing its points, lines and polygons respectively.

  ArcToolbox

  ```
  Export from TIN to coverage: Points ⇒ tin2points,
                               Lines ⇒ tin2lines,
                               Polygons ⇒ tin2polygons.
  ```

- Classification of TIN facets, and selection of respective polygonal region boundaries according to the slope threshold chosen.

  ArcMap

  ```
  Select from tin2lines arcs where:
  {{LSLOPE ≤ threshold and LSLOPE ≥ -threshold} and {RSLOPE > threshold or
  RSLOPE < -threshold}} or
  {{RSLOPE ≤ threshold and RSLOPE ≥ -threshold} and {LSLOPE > threshold or
  LSLOPE < -threshold}} or
  LSLOPE = -9999 or
  RSLOPE = -9999;
  ```

  where: LSLOPE ("left polygon slope") and RSLOPE ("right polygon slope") are standard fields of the AAT info files;
  A slope value of -9999 represents the universe polygon.

  Create shapefile with selected arcs: *#arcs.shp.*

  ArcToolbox

  Export shapefile to coverage and create polygon topology:

```
Export #arcs.shp to arcs# coverage;
Build polygon coverage arcs#
```

- Get TIN facets slope values into *arcs#*'s Polygon Attribute Table (PAT), *arcs#*.pat.

ArcToolbox

```
Overlay (UNION) coverage arcs# and tin2polygons ⇒ union#
```

ArcMap

```
Add new item SLOPECLASS to union#'s PAT info file;
Select records from union#.pat whose DEGREE_SLOPE ≤ threshold ⇒ SLOPECLASS
= 0
Select records from union#.pat whose DEGREE_SLOPE > threshold ⇒ SLOPECLASS
= 1
```

ARC

Extract items *arcs#*_ID (standard field where the object ID is stored) and SLOPECLASS from *union#*.pat ⇒ create with them a new info table *extfromunion#*:

```
PULLITEMS union#.pat extfromunion# arcs#_ID  SLOPECLASS
```

ArcMap

```
JOIN table arcs#.pat and table extfromunion#; (vd. Figure 5.14)
```

Colour code coverage *arcs#* using SLOPECLASS field values (*vd.* Figure 5.12).

Figure 5.14 below shows an example of a PAT info file with the resulting binary classification of the TIN facets for the *arcs45* coverage (Kew dataset): records whose DEGREE_SLOPE ≤ 45° ⇒ SLOPECLASS = 0; DEGREE_SLOPE > 45° ⇒ SLOPECLASS = 1.

Attributes of arcs45 polygon

Y-CENTER2	AREA2	PERIM2	SLOPECLASS
177314.484375	27.054199	131.770873	1
177156.015118	34305.215820	5177.044980	0
177311.045793	23.373535	41.120015	1
177271.988604	2111.659668	1126.497381	1
177312.095410	98.002197	102.985485	1
177314.28125	15.380615	65.273356	0
177313.598958	5.726074	28.886069	0
177312.746191	26.805664	26.073792	0
177312.974071	33.802002	35.615680	0
177313.507479	15.424072	53.733798	1
177304.437418	188.178711	134.651652	1
177308.339131	65.141113	56.240387	1
177309.123156	35.558350	48.255226	1
177191.695995	4150.387207	2026.755840	1
177309.973004	15.771729	16.042557	0
177312.053922	6.616943	13.147807	0
177312.453125	1.624512	7.411190	0
177310.854167	4.854980	10.185419	0
177311.644726	7.697510	13.691112	0
177304.941483	118.548828	70.436952	0

Figure 5.14 – A PAT info file showing the binary classification of the TIN facets: "flat" polygons (SLOPECLASS = 0), and "steep" polygons (SLOPECLASS = 1).

5.5 Summary

For the purposes of development and test of the graph-based technique conceived in this project, LiDAR data was used as the test environment. Thus, this chapter started by reviewing the principles of LiDAR systems, the data and its main applications.

The initially unstructured geospatial datasets of urban areas used in this research, and some of their technical specifications, were presented (both the LiDAR as well as the photogrammetric datasets).

Starting from the datasets mentioned above, this chapter also covered the practical methods performed in pre-processing the raw data. In order to start structuring these data, it was explained in particular how topology was made more explicit by describing the internal shape of the point sets. This was accomplished by generating a TIN across them, based on the Delaunay triangulation. Then, a binary classification of the TIN facets based upon their gradients was employed ("flat" and "steep" facets), and the thresholding methodology used for this purpose was presented in detail. Eventually, the TIN facets were "aggregated" according to the classification previously performed, in order to generate a map of polygon gradient regions. It was also pointed out how the steep polygonal regions in particular are expected to enclose the most important urban features.

After describing in this chapter how the raw data were pre-processed, next chapter covers the conceptual considerations based on graph theory taken into account in designing a technique for the analysis and visualisation of urban topology. Chapter 6 starts by showing how the resulting maps of gradient regions were used as a starting point in the urban topology analysis process.

6 A containment-first search algorithm

The test environment and the preparation of the data used were presented in the previous chapter. In this chapter, the conceptual foundations of the development of a tool for analysis and visualisation of spatial topology are outlined. Also covered in this chapter are the technical aspects of the implementation of such a tool.

This chapter is structured as follows. Given the manner in which the unstructured data were pre-processed, the conceptual considerations of how graph theory was applied in the design of such a tool are presented in section 6.1. Details of the construction of the network of connectivity, throughout the pre-processed collection of spatial objects, are given in section 6.2 - in particular, the retrieval of polygon adjacency information and also the retrieval of the spatial relations of touching between steep polygons; aspects of how to represent graphs within a computer are also discussed. The foundations of the analytical method for spatial topology analysis are described in section 6.3: for example, the initial thoughts based on visual inspections of the graph of polygon adjacencies; the interpretation of the urban topology based on the extension of the standard notion of the spatial relation of adjacency to that of containment; the translation of this interpretation into something that can be visualised in the GIS environment (*containment-first search* and *rationalization* procedures). The first experiments carried out with the case study area of the London dataset are also described in section 6.3; the limitations of the technique that became evident from the initial experiments are indicated, and the process undertaken to extend the *containment-first search* procedure in order to address the limitations - by detecting *polygon-ring containments* - is also described.

Section 6.3 ends by coding the algorithm. An interactive tool was also developed for the visualisation of the urban spatial topology; its implementation is described in section 6.4. Finally, a summary of this chapter is given in section 6.5.

The diagram depicted in Figure 6.1 illustrates the processing and data flows in the urban spatial topology analysis.

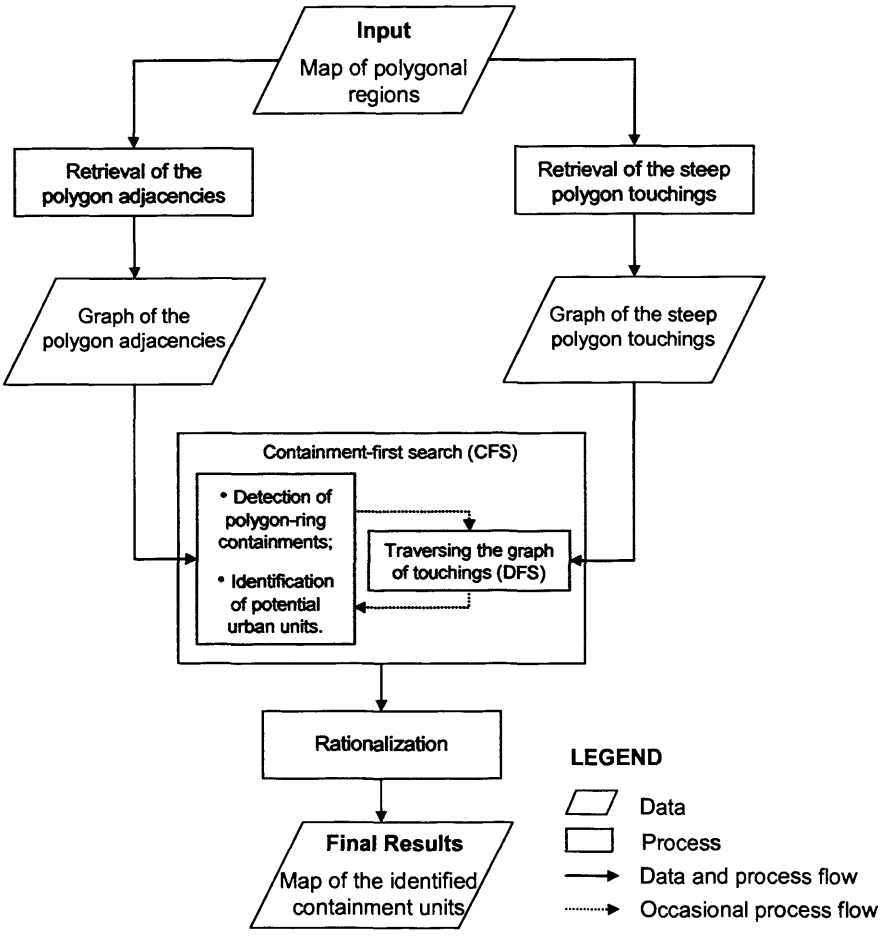


Figure 6.1 – An overview of the main steps of the graph-theoretic algorithm for the analysis of spatial topology.

The computer program developed was implemented in ANSI C (*vd.* associate code in Appendix E). This was principally because of the potential advantages of using pointer variables in C (Kernighan and Ritchie, 1988; Kelley and Pohl, 1990), especially in dealing with graph processing. A pointer is just a variable that contains the address of an object in memory. Pointers are much used in C, partly because they are sometimes the only way to express a computation, and partly because they usually lead to more compact and efficient code than can be obtained in other ways

(Kernighan and Ritchie, 1988). Unlikely most programming languages, C provides for pointer arithmetic; pointer arithmetic is considered one of the strongest aspects of the language (Kelley and Pohl, 1990).

6.1 A network of connectivity across the polygonal regions

After the preliminary task of processing the raw data (comprising the creation of a TIN, the binary classification of the TIN facets, and the generation of the polygonal regions), the next step is to build up the network of connectivity throughout the map of polygonal regions by applying graph theory, as described below.

Figure 6.2 illustrates the way this network of connectivity was conceived: each region is represented by one vertex in the graph, and graph edges link up graph vertices according to the polygons' adjacencies.

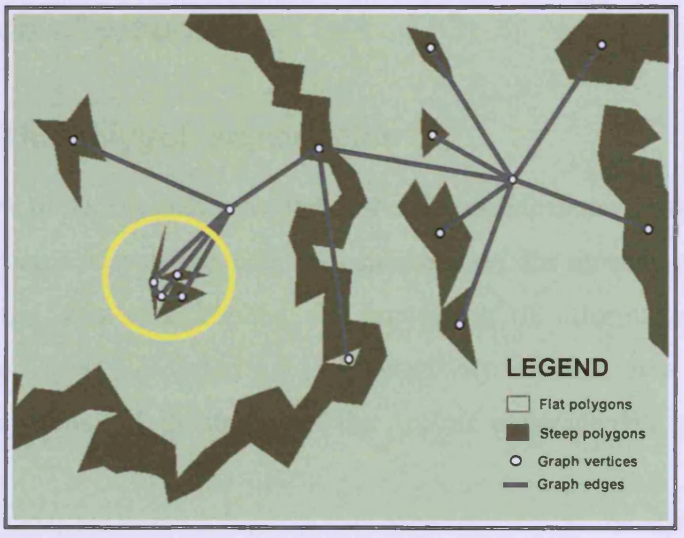


Figure 6.2 – Establishing a network of connectivity between polygonal regions.

Given the way the network of connectivity between polygonal regions was conceived, it is necessary to know which polygon in the map is adjacent to which. This information is available from the topological data structure (*vd.* Chapter 2, section 2.6).

As discussed in section 2.3, and for the purpose of this work, the spatial relationship of *adjacency* between polygons is defined as follows: two polygons are adjacent if and only if they share at least one arc. As far as the spatial relationship of

containment is concerned, a broader understanding than the usual notion of containment between polygons is adopted in this research. It is clear that a polygon is contained by another polygon if the former is completely surrounded by the latter. However, in our case it is also possible to have a polygon surrounded by a ring of two or more polygons of the same class, which have not been merged into one single entity because they happen to meet only at nodes. In this case, the former polygon is said to be *contained* by this ring of polygons. Figure 6.2 also depicts an example of this particular case of the spatial relationship of containment: a flat polygon can be seen on the left of the diagram contained by a ring of three steep polygons (highlighted by the yellow circle). If two polygons happen to meet at a node, these are not said to be adjacent. In this case, the spatial relation is called *touching* (*vd.* Chapter 2, section 2.3, in order to see how these topological terms relate to the accepted literature on spatial topology).

6.2 Construction of graphs

6.2.1 Retrieval of the polygon adjacencies

It was necessary to access polygons and associate arc attributes in order to retrieve polygon adjacencies. Given the GIS data model used for storing vector data, the ArcInfo coverage, this task implied the processing of information spread over various lists and info files referring to connectivity of arcs, area definition and contiguity of polygons (*vd.* illustration of the ArcInfo coverage data model in Figure 2.16).

This task was carried out automatically, and for this purpose a routine was implemented in the ArcInfo embedded programming language, *Arc Macro Language* (AML). Developing this program in AML had the advantage of using some command scripts that had been implemented and were available in the UCL Department of Geomatic Engineering.

The main operations implemented in the AML routine mentioned above are summarised below in terms of pseudo code (the associate code is included in Appendix A):

- Retrieve and list all the polygons adjacent to each polygon.

  ARC

  The PAL file (Polygon-arc list, *vd.* Figure 2.16) is not directly accessible. PALINFO command is used to convert either a polygon PAL file or region PAL file into an info file which can be accessed:

  `PALINFO arcs# ⇒ arcs#_adjacencies,`

  where *arcs#* is the input coverage, and *arcs#_adjacencies* is the output info file.

  Pull fields UNIT (where the polygon IDs are stored) and ADJACENT (where the adjacent polygon IDs are stored) from *arcs#_adjacencies*, in order to create a new info file, *adjacencies*, containing only those two fields:

  `PULLITEMS arcs#_adjacencies adjacencies UNIT ADJACENT.`

- Take the info file *adjacencies* obtained, sort this table by field UNIT and by field ADJACENT; delete repeated rows.

  ARC

  `&RUN filteradjacencies` (a sub-routine to sort tables by specified fields, and delete repeated rows; from Ayugi, 2006):

  ```
  TABLES
  SELECT adjacencies
  SORT UNIT ADJACENT
  ARC
  Loop through the sorted table adjacencies
  Take one row at a time; compare it with the next one
      If they are equal Then
         Delete the second one
      End if
  Proceed down to the end of the table adjacencies
  End loop
  ```

- Read *adjacencies'* rows (UNIT and ADJACENT fields) and write them into a text file, *outputadj#.dat.*

The output of this procedure is a text file, *outputadj#.dat*, which is basically a list translating the graph edges: each line of the file contains a pair of adjacent polygons, represented by their ID, which defines an edge of the graph. This was the format used to define the graph which, in turn, constitutes the input of the analysis program.

As an example, there is a print out of such a file in Appendix B for the case study area of the London dataset.

After creating the graph of adjacencies for each dataset used, these were visualised in order to validate correct graph creation. As an example, the graph of adjacencies obtained for the case study area taken from the London (Kew) dataset is drawn in Figure 6.3. It is important to mention that a considerable amount of time was spent searching for suitable graph drawing software, through the World Wide Web and also talking to different experts in the field. Several packages were considered, such as GTL/Graphlet, LEDA/AGD, GDToolKit, Graphviz, daVinci, and drawGraph.m for MATLAB. For several reasons, mainly technical and licence issues, it was not possible to run and try most of them; the exception was Graphviz, which did not suit our purposes. This tool cannot deal efficiently with large graphs comprising, for example, hundreds of vertices. Eventually, UCINET 6 for Windows (Borgatti *et al.*, 2002) was selected; this graph drawing software is able to represent large graphs, but does not provide a planar layout for planar graphs.

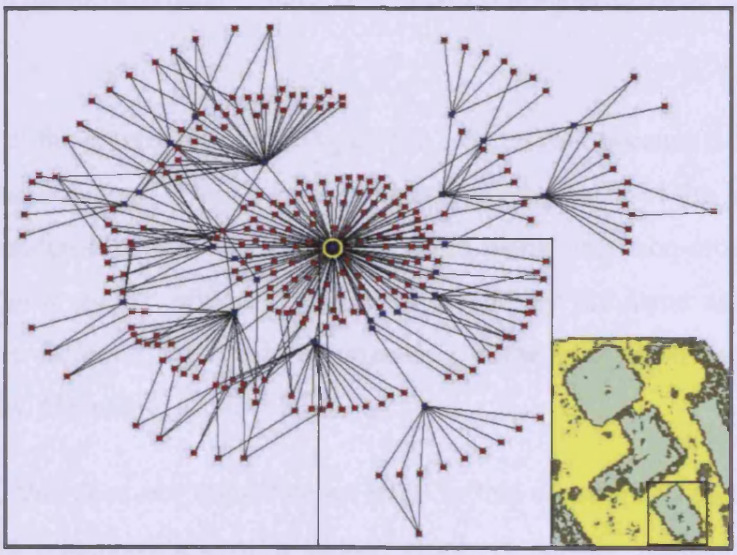


Figure 6.3 – Graph of adjacencies for the London dataset's case study area (generated using Ucinet 6 for Windows; Borgatti et al., *2002); boxes refer to the areas enlarged in Figure 6.9. (From de Almeida* et al., *in press)*

The original map of polygonal regions is a planar graph in itself because of the planar enforcement discussed in Chapter 2: the polygon arcs represent the graph edges; and the arc intersections (nodes) represent the graph vertices. Given the way it was conceived, the associate graph of polygon adjacencies is the dual graph of that

(and *vice versa*, *vd.* Chapter 3, section 3.3). Finally, it is known that the dual of a planar graph is a planar graph too, as "every planar graph has a (planar, combinatorial) dual" (Gibbons 1989, theorem 3.8, pg. 83). Therefore, it is known beforehand that the derived graphs of polygon adjacencies, like the one depicted in Figure 6.3, are planar graphs.

As far as the visual representation of graphs is concerned, two options were considered: planar and non-planar layout. In theory "a planar graph can be drawn on the plane without any crossing edges, and such that no two end-nodes coincide" (Wilson, 1996). However, this fact is not actually evident in the graph layout pictured in Figure 6.3, as several crossing edges can be spotted across the diagram. In practice, drawing a planar graph with a layout containing no crossing edges is an awkward task, and most of the software packages available do not provide this functionality. Moreover, with the obvious exception of the graphs containing loops or multiple edges, "every planar simple graph can be drawn in such a way that all its edges are represented by straight lines" so that no edges intersect (Chartrand and Lesniak, 1986, cited in Wilson, 1996), as proved by Wagner in 1936 and by Fáry in 1948.

The position of the graph vertices in Figure 6.3 is irrelevant because it does not have any geographic meaning. Thus, a different spatial arrangement of the graph vertices could be considered in order to draw the graph using only non-crossing straight edges. The "new graph" obtained would be essentially the same as the original, though with a different layout. More precisely, these two graphs are said to be isomorphic (*vd.* Chapter 3, section 3.3).

Nevertheless, this does not constitute an issue in this research. In fact, in our case either graphic representation of a planar graph is equally acceptable. What is pertinent to point out is that in both cases the relative position of the vertices, in relation to each other, would remain intact (*i.e.* their topological relationships would be preserved).

6.2.2 Retrieval of the steep-polygon touchings

The information required for the retrieval of the steep-polygon touchings is not explicit in the database either. As for the retrieval of the polygon adjacencies, a complex task, entailing the manipulation and analysis of information spread over different INFO files (Chapter 2, section 2.6), had to be carried out. The results were extracted from the Arc Attribute Table (feature attribute table storing topology information for arcs of either a polygon coverage or a line coverage), and from the Polygon Attribute Table (feature attribute table storing topology information for arcs of a polygon coverage) (ESRI, 1995; Rigaux *et al.*, 2002; ESRI, 2005).

In order to perform this task automatically, a routine was also implemented in *Arc Macro Language* (AML) to access the required information, and hence to retrieve steep-polygon touchings. The main operations implemented in this routine are summarised below in terms of pseudo code (the associate code and interim results of the various operations are attached in Appendix C):

- Creation of two tables T1 and T2 with columns FNODE# (from-node ID), TNODE# (to-node ID), LPOLY# (left-polygon ID) and RPOLY# (right-polygon ID) from the *arcs#*.aat info file.

```
ARC

PULLITEMS arcs#.aat T1 (vs. T2) FNODE# TNODE# LPOLY# RPOLY#
```

- Declaration of two cursors, curT1 and curT2, to read respectively both tables T1 and T2 ⇒ copy rows from T1 in reversed order into T2.
- Creation of table T3 with columns FNODE# and LPOLY# from T2.

```
ARC

PULLITEMS T2 T3 FNODE# LPOLY#
```

- Sort rows of T3 by FNODE#, and by LPOLY#; delete repeated rows (from Ayugi, 2006):

```
TABLES

SELECT T3
SORT FNODE# LPOLY#

ARC

Loop through the sorted table T3
    Take one row at a time; compare it with the next one
        If they are equal Then
            Delete the second one
```

```
        End if
    Proceed down to the end of the table T3
End loop
```

- Creation of a new table T4 with columns *arcs#*_ID (where polygon ID is stored) and SLOPECLASS from the *arcs#*.pat info file.

```
ARC

PULLITEMS arcs#.pat T4 arcs#_ID SLOPECLASS
```

- Rename columns *arcs#*_ID to PolyID in table T4; both FNODE# and LPOLY# in table T3, to NODE# and PolyID respectively.

```
TABLES

SELECT T4
ALTER arcs#_ID PolyID
SELECT T3
ALTER FNODE # NODE#
ALTER LPOLY # PolyID
```

- Creation of a new table T5 by joining tables T3 and T4 through PolyID column.

```
ARC

JOINITEM T3 T4 T5 PolyID
```

Select rows from T5 where: SLOPECLASS = 0 (flat polygons) ⇒ delete them:

```
TABLES

SELECT T5
RESELECT SLOPECLASS = 0
PURGE
```

Also delete column SLOPECLASS from T5:

```
TABLES

DROPITEM T5 T5 SLOPECLASS
```

Now, T5 has columns NODE# and PolyID.

- Create an auxiliary table T5_2 from T5 and rename PolyID column to PolyID_2.
- Creation of a new table T6 by joining tables T5 and T5_2 through NODE# column.
- All occurrences from both T5 and T5_2 are combined into T6.

```
Declare cursors, curT5, curT5_2 and curT6, to read tables T5, T5_2 and T6
respectively.
Loop through T5
    Loop through T5_2
        If NODE# in T5 = NODE# in T5_2 Then
```

```
            Write NODE#, PolyID from T5, PolyID_2 from T5_2, into T6
        End if
    End loop
End loop
```

- Deletion of rows in T6 where a polygon shares a node with itself, and with the universe polygon.

```
TABLES
SELECT T6
RESELECT PolyID = PolyID_2
PURGE
RESELECT PolyID = -9999
PURGE
```

- Sort rows in T6 by PolyID, and by PolyID_2; delete repeated rows (as explained above).
- Create and open the output file; with the cursor pointer, read table T6 and copy each pair of polygon IDs, PolyID# & PolyID_2#, into the output file.

The output of this procedure is a text file, *outputnodeshare#.dat*, containing in each line a pair of touching steep polygons, represented by their IDs. Likewise for the polygon adjacencies, this output file is basically a list of the edges of another graph, called the *graph of steep-polygon touchings*. As an example, there is a print out of such a file in Appendix D for the case study area of the London dataset.

After obtaining the spatial relations of touching between steep polygons, the respective graph of touchings was constructed for each dataset used. As an example, Figure 6.4 shows the graph of touchings obtained for the case study area taken from the London (Kew) dataset.

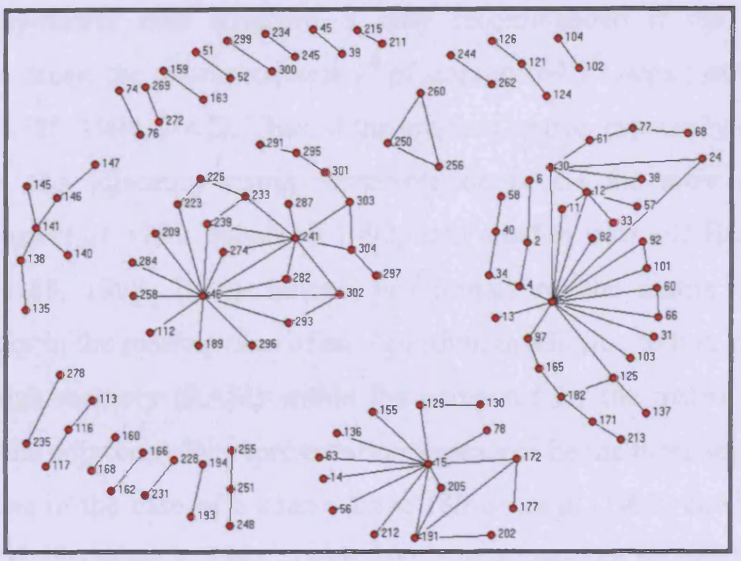


Figure 6.4 – Graph of touchings between steep polygons for the London dataset's case study area (generated using Ucinet 6 for Windows; Borgatti et al.*, 2002).*

As can be observed in the figure above, the graph of touchings obtained is a disconnected graph, a characteristic which was expected *a priori*. Indeed, not all the steep polygons are in direct contact to one another by a shared node. For the same reason, isolated steep polygons are not part of this graph.

6.2.3 Graph data structures

6.2.3.1 Adjacency-matrix *vs.* adjacency-list representations

In order to process a graph with a computer program, it needs to be represented within the computer by using an appropriate data structure. There are two common methods used to represent graphs in a computer (Sedgewick, 1988, 1998, 2002): the *adjacency-matrix* representation; and the *adjacency-list* representation. These graph data structures are both arrays of simpler data structures, one for each vertex, describing the edges incident on that particular vertex (Sedgewick, 1998).

The adjacency-matrix representation is the most straightforward data structure and is implemented as an indexed two-dimensional array, V-by-V (V represents the total number of vertices), of boolean values: $A = [a_{ij}]$, where a_{ij} is set to *true* if there is an edge linking up v_i and v_j, and otherwise to *false* (Kruse *et al.* 1991, Schalkoff 1992, both cited in Barr and Barnsley, 1997; Sedgewick, 1988, 1998, 2002). This approach has already been presented and illustrated in Chapter 3.

The adjacency-matrix data structure is only recommended if the graph to be represented is dense: the matrix requires V^2 of storage and V^2 steps just to initialise it (Sedgewick, 1988, 1998, 2002). Thus, if the graph is sparse, especially if it is a huge sparse graph, the adjacency-matrix representation is not the most suitable data structure (Kruse *et al.* 1991, Schalkoff 1992, both cited in Barr and Barnsley, 1997; Sedgewick, 1988, 1998, 2002). Indeed, just initialising the matrix could be the dominant factor in the running time of an algorithm; in addition to this, there may not even be enough memory (RAM) within the computer for the matrix (Sedgewick, 2002). Thus, the adjacency-list representation appears to be the most appropriate data structure to use in the case of a sparse graph (Kruse *et al.* 1991, cited in Barr and Barnsley, 1997; Sedgewick, 1988, 1998, 2002). It allows the processing of all the edges of a graph in time proportional to $V+E$, *i.e.* the total number of vertices plus the number of edges, as opposed to V^2 (Sedgewick, 1998, 2002).

Because the graphs of adjacencies obtained in this project may well have hundreds of vertices (*e.g.* one of the graphs obtained for the Stuttgart's Site 1 dataset is a 9415-vertex graph, *vd.* Chapter 8), and are fairly sparse (the vast majority of the vertices have a low valence), the adjacency-list structure was chosen as the graph data structure.

Sedgewick (1988, 2002) also emphasised another important aspect of this representation, well worthy of being taken into account - "the order in which the edges appear in the input is quite important: it (along with the insertion method used) determines the order in which the vertices appear on the adjacency lists". A consequence of this is that a graph can be represented in many different ways in an adjacency-list structure, depending upon how its edges are ordered in the input. Given the GIS software package used in this research, the polygon adjacencies are retrieved in such a way that the associate graph edges are sorted in the input file by "from-vertex" index ascending, then by "to-vertex" index ascending (*vd.* example in Appendix B). This fact, along with the insertion method used (which inserts the edge connecting vertices v and u by placing v at the top of u's adjacency list, and u at the top of v's adjacency list), results in an array of linked lists sorted in descending order of vertices IDs.

As Sedgewick (1988) noted, "the order in which edges appear on the adjacency lists affects, in turn, the order in which edges are processed by algorithms". An example of this fact is the order in which the graph edges are traversed, which is inevitably affected; indeed, this is evident enough if we make a visual inspection of a resulting spanning tree. Let us consider again the simulated graph given in Chapter 3, section 3.4 (*vd.* Figure 3.6). The associate DFS and BFS traversal trees, pictured again in Figure 6.6a) & b), were drawn for illustration purposes assuming that the graph was represented by an array of linked lists sorted in ascending order (*vd.* Figure 6.5a). If we reverse the sorting of these lists, *i.e.* descending order (like the actual ones we obtained - *vd.* example in Figure 6.5b), the visual configuration of both DFS and BFS traversal trees would be like those depicted in Figure 6.6c) & d), respectively.

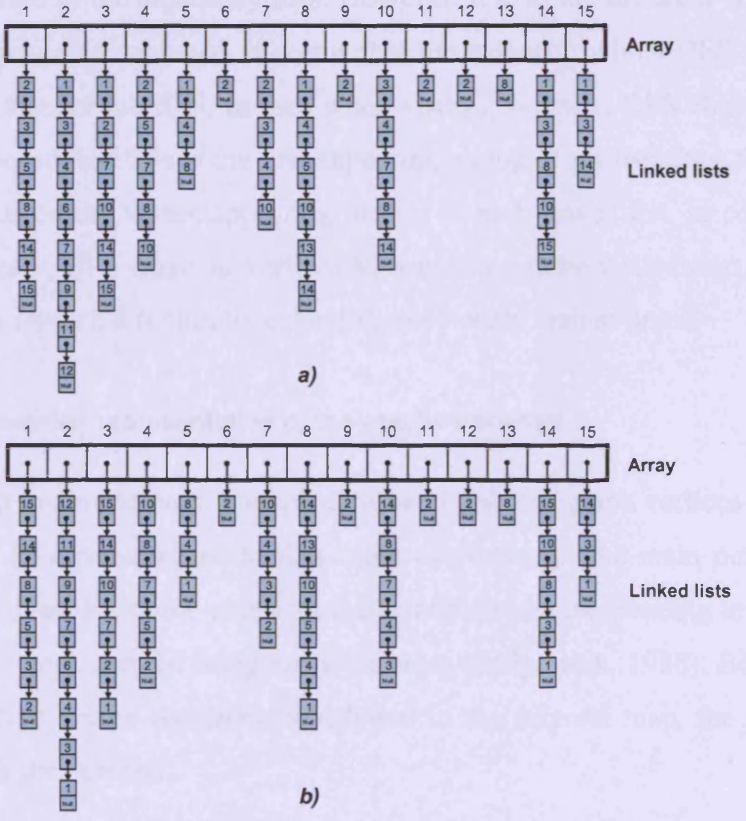


Figure 6.5 – Two different adjacency-list representations of the simulated graph in Figure 3.6: a) linked lists sorted by vertex index ascending; b) linked lists sorted by vertex index descending. (After Kelley and Pohl, 1990; Sedgwick, 1998)

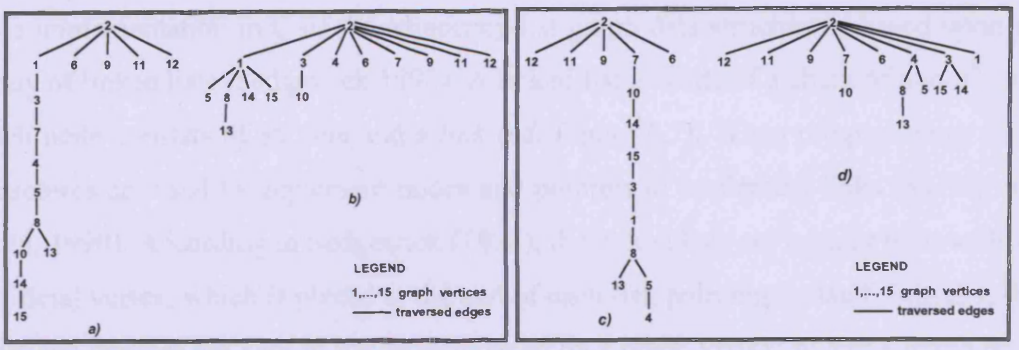


Figure 6.6 –Spanning trees for the simulated graph in Figure 3.6: a) DFS tree – graph stored using adjacency-list representation with linked lists sorted by vertex index ascending; b) BFS tree - vertex index ascending; c) DFS tree – vertex index descending; d) BFS tree - vertex index descending.

As Figure 6.6 shows, the configuration of the spanning trees obtained by DFS and BFS algorithms is different in both cases, depending on the order in which the graph edges are sorted in the adjacency lists. However, due to the different ways in which both algorithms are conceived, it seems that the extent in which DFS is affected is greater then that for the BFS. In fact, when visiting vertex v, DFS chooses only one vertex to proceed, and this is the first appearing on top of v's list; thus, the path to be taken depends on the vertex appearing on top of each linked list. In contrast, when visiting vertex v, BFS takes all vertices adjacent to v to be visited next, irrespective of their order in v's list (naturally excluding previously visited links).

6.2.3.2 The adjacency-list representation of the graphs obtained

The first step in structuring the graph data, is to give the graph vertices numeric IDs from 1 to V (V represents the total number of vertices). The main purpose of this procedure is to enable quick access to the information corresponding to each vertex, which can be implemented using array indexing (Sedgewick, 1988). Because in our case each graph vertex represents a polygon in the original map, the polygon IDs were used for the purpose.

As mentioned in section 6.2.1, the text file containing the graph edges constitutes the input of the graph analysis program. This program creates the adjacency graph by reading in its edges and, simultaneously, by assigning to each vertex an integer:

$$\text{vertex } i \text{ read} \xleftarrow{\text{assigned}} \text{integer } i$$

The implementation in C of the adjacency-list graph data structure is based upon an array of linked lists (Sedgewick 1998). A linked list is made of a chain of *nodes*[9], and each node consists of an *item* and a *link* (*vd.* Figure 6.7). When programming in C, structures are used to implement nodes and pointers to implement links (Kelley and Pohl, 1990). According to Sedgewick (1988), the linked lists are usually built with an artificial vertex, which is placed at the end of each list, pointing to itself. Initially, the artificial vertices are kept in the array indexed by a graph vertex; to add a graph edge connecting, for instance, vertices x and y, x is added to y's adjacency list and y to x's adjacency list. As can be seen, the edge x-y is represented twice: it is represented as a node containing x in y's adjacency list, and as a node containing y in x's adjacency list, *i.e.* a set of bidirectional edges. Sedgewick (1988, 2002) stated that this "double representation"-based implementation is necessary, otherwise it would not be possible to extract efficiently certain details of the graph structure, such as which vertices are connected directly to vertex v.

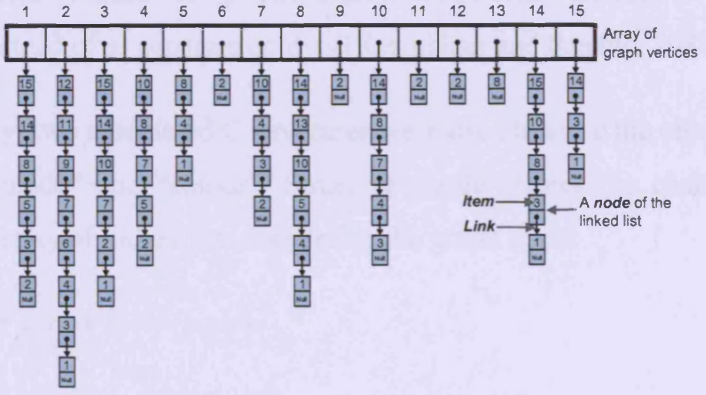


Figure 6.7 – The adjacency-list representation of a graph within a computer. (Adapted from Sedgewick, 1998)

For illustration purposes, let us consider again the simulated graph given in Chapter 3, section 3.4. As an example, let us take two edges of that graph, *1-2* and *1-3*, which are read in this order by the computer program. Figure 6.8 illustrates how these edges

[9] In the context of linked lists, a *node* is an element of the chain that builds up a linked list. It comprises two members: the member *data*, which stores the index of that particular list element (item); the member *next*, which stores the link to the successor list element (this link is implemented by using a pointer variable containing the address of the successor list element) (Kelley and Pohl, 1990).

and associate nodes are sequentially added to the adjacency lists. Figure 6.7 depicts the complete adjacency-list representation of the simulated graph mentioned above.

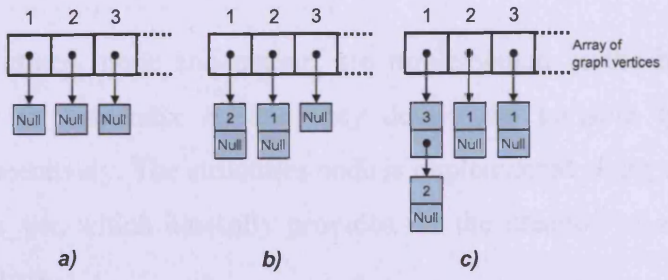


Figure 6.8 – Adding edges to a graph represented in a computer by adjacency lists: a) initial stage; b) edge 1-2 is added; c) edge 1-3 is added.

As explained above, the adjacency-list graph data structure was implemented by using C structures, combined with a Typedef type, and by using pointers. A structure in C is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. C structures help to organise complicated data because they permit a group of related variables to be treated as a named unit instead of as separate entities (Kernighan and Ritchie, 1988).

More precisely, two associated C structures were used to store the obtained graphs of adjacencies: "node" and "colour". Structure "node" stores the chain of links and builds up the array of linked lists that define the graph itself:

```
typedef struct node *a_link;
struct node
{int v; // v = item
 a_link next;} // next = link
```

Structure "colour" stores the graph vertex attributes, namely: the index of the unit of containments that the respective polygon belongs to, containment_unit; the level of containment within the given containment unit that the respective polygon belongs to, containment_level; the valence of the vertex, valence; the parent of the vertex in the traversal process, parent; the colour filling style of the respective polygon to be used in mapping it on the original map, FillStyle; and the three components to define the colour to be used for this purpose – Sue (H), Saturation (S), and Value (V),

```
typedef struct colour node_colours;
struct colour
{int containment_level;
 int containment_unit;
 int valence;
```

```
    int parent;
    int FillStyle;
    int H;
    int S;
    int V;}
```

These two structures, node and colour, are implemented at the beginning of the program listed in Appendix E, and they define the variable types a_link and node_colour, respectively. The structures node is implemented along with the function NEW, of type a_link, which basically provides for the creation of a new link in an array of linked lists.

Two different arrays of linked lists were defined to store both the graph of polygon adjacencies and the graph of steep-polygon touchings: adj[] and touch[], respectively. Algorithm 6.1 populates adj[] by reading in the adjacency graph edges from the input file *outputadj#.dat*, and by constructing the associate array of linked lists (*vd.* Function 1 in Figure 6.22, and in Appendix E). Similarly, Algorithm 6.2 reads in the edges of the steep-polygon touching graph, from the input file *outputnodeshare#.dat*, and populates touch[] (*vd.* Function 2 in Figure 6.1, and in Appendix E).

INIT_adj Algorithm 6.1

```
//INPUT: outputadj#.dat file (fixed name);
//Initialisation of the array of linked lists adj[];
//Open/read input file of graph adjacencies;
While not end of the input file Do
    //Take one line at a time: read first vertex s, and second vertex t;
    If the edge s-t isn't represented twice in the file (s-t vs. t-s)
    Then read it in
        //Count number of "new" lines;
        adj[s] = new link to t;
        adj[t] = new link to s;
    End if
    //Move on to the next line;
End while
Nbr of graph edges = Nbr of "new" lines;
//Close input file;
End
//OUTPUT: array of linked lists adj[].
```

INIT_touch Algorithm 6.2

```
//INPUT: outputnodeshare.dat file (fixed name);
//Initialisation of the array of linked lists touch[];
//Open/read input file of graph touchings;
While not end of the input file Do
    //Take one line at a time: read first vertex u, and second vertex v;
    If the edge u-v isn't represented twice in the file (u-v vs. v-u)
    Then read it in
        //Count number of "new" lines;
        adj[v] = new link to u;
        adj[u] = new link to v;
    End if
    //Move on to the next line;
End while
Nbr of graph edges = Nbr of "new" lines;
```

```
//Close input file;
End
//OUTPUT: array of linked lists touch[].
```

6.3 The method for spatial topology analysis

6.3.1 Preliminaries

As can be seen in Figure 6.3, polygon *3* (highlighted in yellow on the bottom right) is the most connected flat polygon, and in this case it is also the largest in terms of area. This polygon constitutes the *Useful External Border* (*vd.* concept of *UEB*, Nardinocchi *et al.*, 2003; Forlani *et al.*, 2006) with which the graph drawing was started (the corresponding vertex in the graph is also highlighted in yellow).

In a geometric and topological description of regions, Nardinocchi *et al.* (2003) introduced the concept of the UEB of a region, which the authors define in their research as a quantitative measure: "the number of pixels that surround (do not belong to) the region and exist (do not belong to the image external border)".

Since the definition above applies to the raster data format, and hence is based on pixels, there is the need to define this concept in a more suitable way for the vector data context. More precisely, and for the purpose of this thesis, the UEB is defined upon a topological measure as: the flat polygon with the highest number of adjacencies, *i.e.* the associate vertex in the adjacencies graph has the highest valence.

The consideration of the concept of UEB in analysing urban topology is a key aspect of the approach proposed in this thesis. The UEB corresponds in practical terms to the terrain polygon, from where sequences of adjacencies in the graph of adjacencies, make most sense in terms of the urban scene (de Almeida *et al.*, in press). In fact, it is a polygon which relates to the others in a special way; in other words, it encloses other polygons referred to potential urban features, such as buildings or vegetation. It is comparable to the *universe polygon* in a GIS data structure.

Now that the UEB has been defined in more precise terms, it is possible to retrieve further geographical information by analysing different paths within the obtained graph of adjacencies. Starting from the UEB, a simple visual observation of the

represented sequences of levels of adjacency between vertices along some paths may tell us that, for instance, a vertex at the end of a path (*i.e.* the highest level of adjacency in that particular path) is a candidate to be either a hole in the ground or something on top of an urban feature (de Almeida *et al.*, in press). This was observed for the test data scenario (LiDAR data); however, it is believed that the concept can be extended to other sources of unstructured data, such as photogrammetric data.

To give an example, let us go through the graph path highlighted in Figure 6.9 (a detail of Figure 6.3). Starting from polygon *3*, at the first level of adjacency the steep polygon *198* is found, which is contained by previous polygon *3*. That, in turn, contains flat polygon *200* at the second level of adjacency. Polygon *200* contains several others and, in particular, contains steep polygons *250*, *256*, *260*, which all together form a ring *containing* flat polygon *257*, belonging to the fourth and last level of adjacency. In terms of urban scene, the meaning of this sequence of spatial relations of adjacency and containment is the existence of a building (pictured on the bottom right of Figure 6.9), whose boundary is shaped by the *rectangular* dark green polygon displayed (de Almeida *et al.*, in press); the shape of this building can also be seen in the associate TIN partially pictured in Figure 6.10.

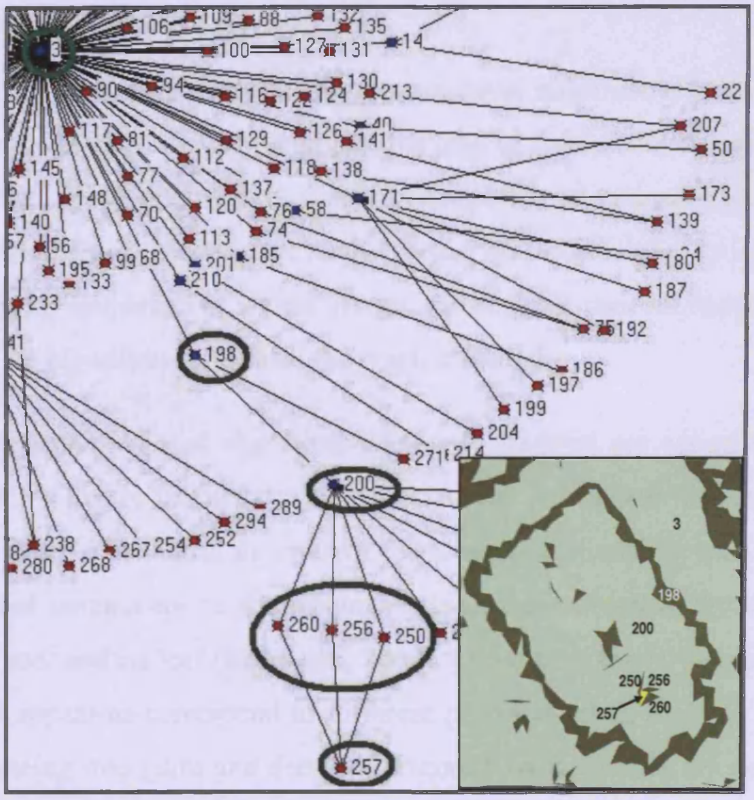


Figure 6.9 – Detail of Figure 6.3: the interpretation of the path highlighted is an example of the geographical information that can be inferred from a graph path, in the context of an urban scene (the numbers on the map are polygon labels in insert).

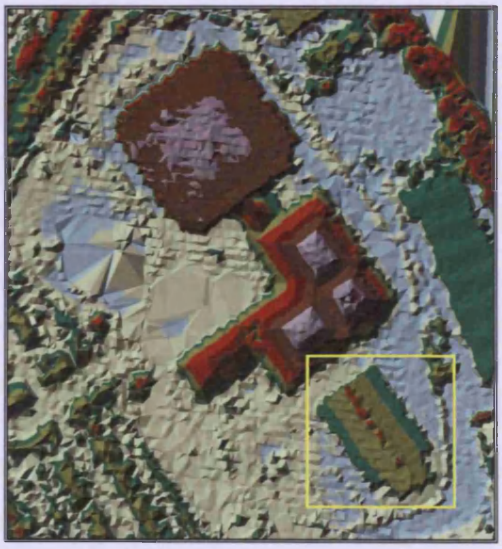


Figure 6.10 – TIN generated for the Kew area (London dataset); the rectangle in yellow highlights the building shaped by the rectangular *dark green polygon in Figure 6.9.*

6.3.2 Containment-first search

As exemplified above, retrieving further geographical information from the graph of adjacencies constitutes the sort of analysis that may be carried out by determining the properties of the graph related to its paths. In Chapter 3, it was noted that this can be done by moving through the graph from vertex to vertex along its edges and by understanding its properties as we go. Hence, the analysis process needs to be based upon one of the algorithms presented for graph traversal.

Given the way the presented algorithms for graph traversal are conceived, it seems that the spanning tree resulting from the BFS traversal (broad and shallow, *vd.* Figure 6.6b & d) is more meaningful in terms of the urban scene: each branch corresponds to a connected component in the original graph; and represents the shortest path between the root and its leaf (Sedgwick, 2002). Looking at the BFS tree's branches, these indeed appear to correspond to different potential urban features. In contrast, the DFS spanning tree (slim and deep, *vd.* Figure 6.6a & c) does not seem to be as easily related to urban features as the previous one, as the interpretation of its long deep path in relation to the urban scene does not appear to be straightforward. Moreover, DFS depends more on vertex sorting in the edge list.

The implementation of the graph analysis procedure was therefore based upon the BFS algorithm. Briefly, the analysis procedure traverses the graph looking for sequential relationships of containment amongst the sequences of adjacency: the *containment-first search* (CFS). The identification of relationships of containment across the whole map of polygonal regions is particularly important, for where containment occurs within the UEB there is a high likelihood of an urban feature being present (de Almeida *et al.*, in press).

The search process can be interpreted as follows. Considering the UEB (a flat polygon) as the starting point of the search, the CFS procedure checks out its adjacent polygons (steep polygons) and assumes each of them as potentially enclosing a different spatial feature. In terms of graph analysis (*vd.* Algorithm 6.3), given a search root: CFS takes the vertices adjacent to the root; each root's adjacent vertex (which belongs to the first level of adjacency/containment) is identified as the beginning of a connected component in the original graph (a new branch in the

associated spanning tree); moreover, each different vertex adjacent to the root is assumed to be potentially enclosing other vertices at higher levels of adjacency (*i.e.* potentially, a new containment unit). Because CFS is based on a graph traversal algorithm, it naturally ignores all the previously traversed links across connected components (branches in the tree), as well as links back to a vertex at a lower level of adjacency within a connected component.

Containment_first_search Algorithm 6.3

```
Graph of adjacencies is traversed (based on breadth-first search), starting
from vertex k:
While Queue of vertices to be visited is not empty Do
    Visit vertex k;
    Start counting the valence of k;
    For each vertex v adjacent to k Do
        Count the valence of k;
        If v hasn't been visited yet Then
            If k is the root Then
                v's parent = k;
                v's level of containment = 1;
                    Nbr of containment units = k's valence;
                v's containment unit = nbr of containment units;
            End if
        Else
            v's containment unit = k's containment unit;
            v's level of containment = k's level of containment + 1;
        Put v in the Queue to be visited thereafter;
    End for
    k's valence = nbr of adjacent vertices v counted above;
End while
Visual representation of the graph analysis results;
End
```

The diagram depicted in Figure 6.11 illustrates how CFS operates particularly in detecting the potential existence of a spatial feature: each vertex adjacent to the root is assumed to be in the basis of a potential spatial feature. Still starting the search process from a flat polygon, say the UEB: the vertex represented in white corresponds to the root; vertices represented with a solid colour correspond to flat polygons; vertices represented with a coloured hashed pattern correspond to steep polygons. In addition, within each tree's branch, the higher the level of adjacency, the darker the tone of the colour assigned to the current branch.

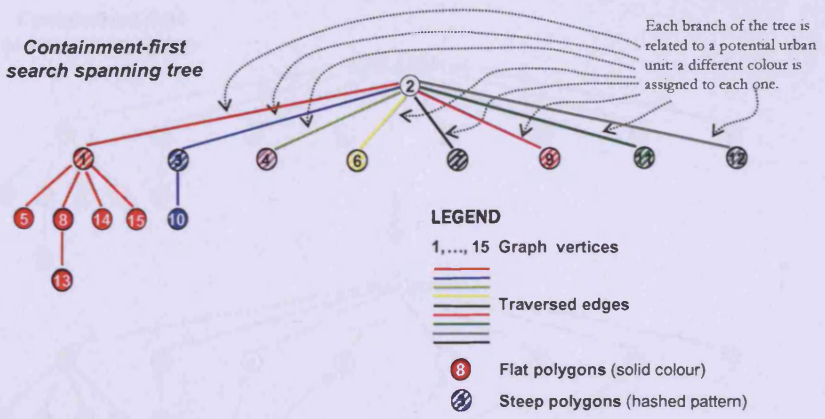


Figure 6.11 – CFS and the detection of potential spatial features: each vertex adjacent to the root is assumed to be in the basis of a potential spatial feature.

6.3.3 Rationalization

A *rationalization* procedure was also implemented to be performed after CFS. One of the aims of the rationalization procedure is to detect and "clear out" steep polygons enclosed by flat polygons. In most cases, steep-polygon islands are related to noise and do not really have any particular meaning in terms of the urban scene, and hence they can be ignored (*vd.* example in Figure 6.12).

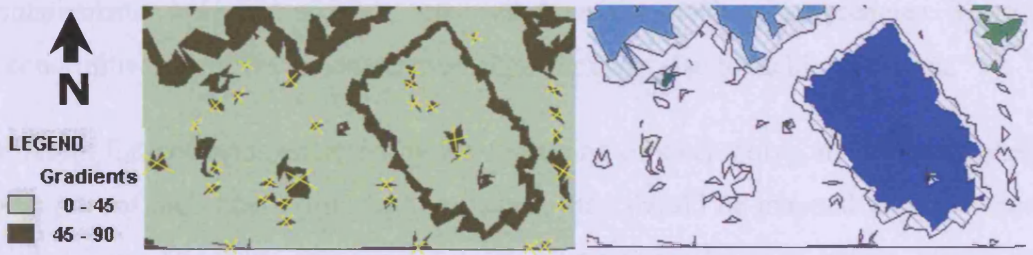


Figure 6.12 – An example of the generalisation procedure: the steep-polygon islands, mostly related to noise, are ignored.

In terms of the algorithm, the rationalization procedure looks for single-valent vertices in the graph (corresponding to leaves in the tree). If a single-valent vertex happens to be a steep polygon, then this is treated as one with its parent (which is an enclosing flat polygon). In terms of the visual representation of this result, the mentioned vertex "inherits" its parent's colour and both are mapped with the same solid colour (*vd.* Figure 6.13). But the "cleared out" vertices are not actually deleted from the original graph, nor from the spanning tree obtained.

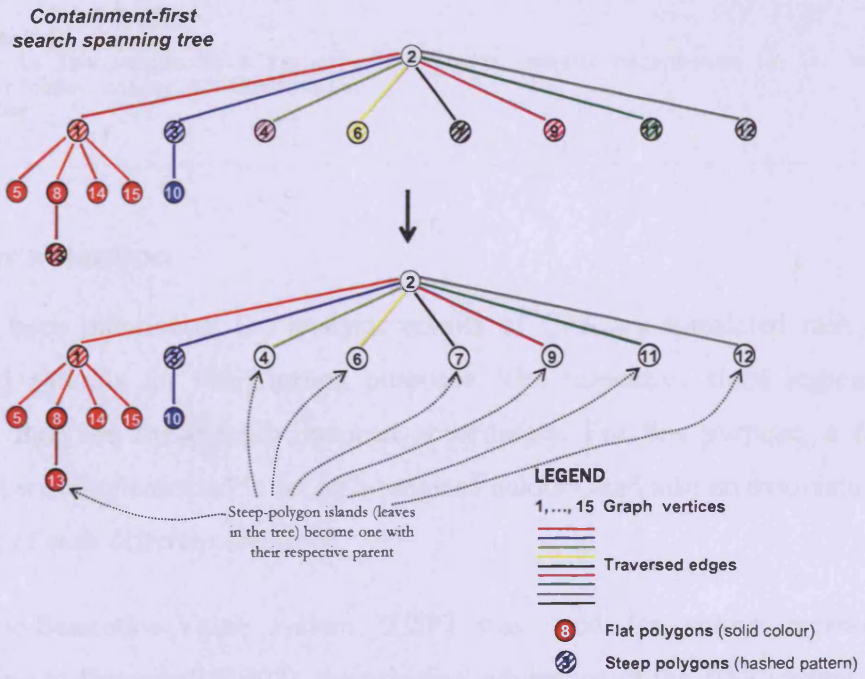


Figure 6.13 – The process of "clearing out" steep-polygon islands within flat polygons.

As shown in section 6.3.5, the first experiments did not produce fully satisfactory results. In fact, by examining the new map of the containment units identified, it was realised that several spatial relations of containment, especially steep-polygon ring containments, were not properly retrieved from the graph of adjacencies. Thus, a second initial aim of the rationalization algorithm was also to tackle this issue.

Different flat polygons enclosed by the same ring of steep polygons are most likely to be part of the same spatial feature; hence, they should be mapped with the same solid colour. The rules developed for the purpose can be seen below, where the rationalization procedure is summarised in terms of pseudo code (Algorithm 6.4).

Rationalization Algorithm 6.4

```
Take the graph of adjacencies:
For each vertex i Do
    If i is single-valent AND represents a steep polygon Then
          Becomes one with its parent
    End if
    If i is bi-valent AND represents a steep polygon Then
          Take i's non-parent adjacent polygon:
          If it belongs to the same containment unit as that of i's parent
          Then Merge i with its parent (in practice, also with its non-parent
          adjacent polygon)
          Else (i's non-parent adjacent polygon belongs to a different
          containment unit)
                Merge i with its parent;
                Merge i's non-parent adjacent polygon with them too;
                If i's non-parent adjacent polygon has steep-polygon islands
                Then Merge them with their surrounding polygon too
                End if
```

```
            End if
        End if
    Write in the output file i's attributes: ID, colour components (H, S, and V),
    colour code, colour filling style;
    End for
    End
```

6.3.4 Colour selection

As has been mentioned, the analysis results of CFS are translated into different coloured patterns for visualisation purposes. The respective slope regions in the original map are dynamically mapped accordingly. For this purpose, a colouring function was implemented to set up a range of colours, and also an associate gradient of tones of each different colour.

The Hue-Saturation-Value system (*HSV*) was used for colour representation. According to Davenhall (1997), the principal advantage of the *HSV* system is that it corresponds closely to the human perception of colour and thus the effects of specifying and manipulating colours in the HSV system are quite intuitive and predictable. Sandwell (2004) also added that, if compared with other systems for colour representation (such as *Red-Blue-Green* system and *Cyan-Magenta-Yellow* system), the *HSV* system provides a more natural way to define a colour.

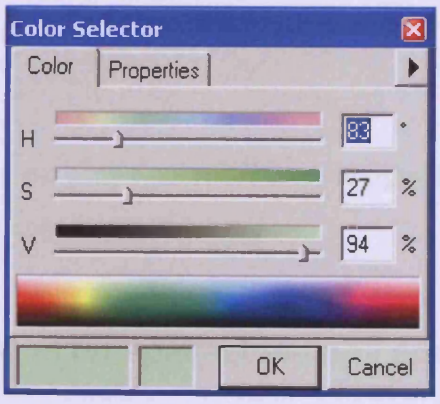


Figure 6.14 – The HSV colour system in ArcMap.

The Hue component, *H*, was mainly used to set up a colour according to the colours of the rainbow – red, orange, yellow, green, blue, violet and back to red (Sandwell, 2004); in terms of numerical values, the colour gamut ranges from 0 to 360° (*vd.* Figure 6.14). Because the idea was to assign a different colour to each containment unit detected, the *H* colour gamut was divided by the number of containment units to define the increment *dH* as follows:

$$dH = \frac{360}{\#\,containment\ units}. \qquad \textit{(Equation 6.13)}$$

Thus, a given particular containment unit will be assigned the colour:

$$H = containment\ unit \times dH. \qquad \textit{(Equation 6.14)}$$

The *HSV* system allows movements in colour space which correspond more closely to what is meant by "tint" and "shade" (Sandwell, 2004). Decreasing S moves the colour toward white, and decreasing V moves the colour toward black. Therefore, given the colour H of a particular containment unit, the Saturation and Value components, S and V, were combined to define together a ramp of tones of that colour. Both S and V range from 0 to 100% (*vd.* Figure 6.14).

An increment dSV was defined as follows:

$$dSV = \frac{200}{\#\,levels\ of\ containment\ of\ a\ given\ unit}. \qquad \textit{(Equation 6.15)}$$

Hence, given a particular containment unit with colour H, and a particular level of containment within the given unit, S and V are calculated as follows:

$$S = Containment\ level \times dSV\ ; \qquad \textit{(Equation 6.16)}$$

$$If\ S \leq 100\ then\ V = 100$$
$$Else\ V = 200 - S$$
$$S = 100.$$

The tones calculated for a particular colour H were assigned to the different levels of containment of the unit mapped with colour H.

6.3.5 Verifying the algorithm for spatial topology analysis

Both CFS and rationalization procedures were applied to the case study of the London (Kew) dataset. The results obtained are pictured in Figure 6.15 below.

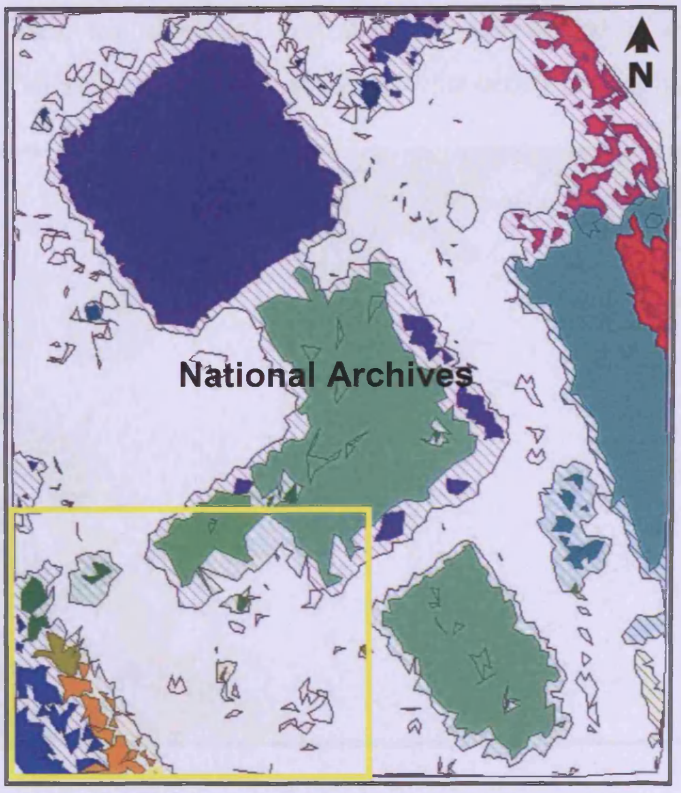


Figure 6.15 – First experiment of the implemented methodology, carried out with the case study area of the London dataset. (Yellow box refers to the area enlarged in Figure 6.16)

Different colours represent disjoint containment units. Solid colours correspond to flat polygons, and coloured hashed patterns correspond to steep polygons. The higher the level of adjacency/containment of polygons within a given unit, the darker the tone of the colour assigned to it.

The idea of assigning a different colour to each containment unit detected during the analysis process, is to help one in visually identifying the potential existence of a spatial feature. When applying the algorithm to geographical data, spatial features may correspond to disjoint buildings (or sets of buildings), or individual trees or sets of trees/bushes.

However, the results obtained with this first experiment do not appear to be meaningful enough. For instance, looking at the National Archives building, it can be seen that more than one colour has unexpectedly been assigned to the different polygons that constitute this containment unit. Another example of the same situation is the southwest part of the map above (*vd.* details in Figure 6.16). By visually

examining that area, the displayed unit seems to correspond to only one spatial feature, and yet different colours were assigned to its constituent polygons.

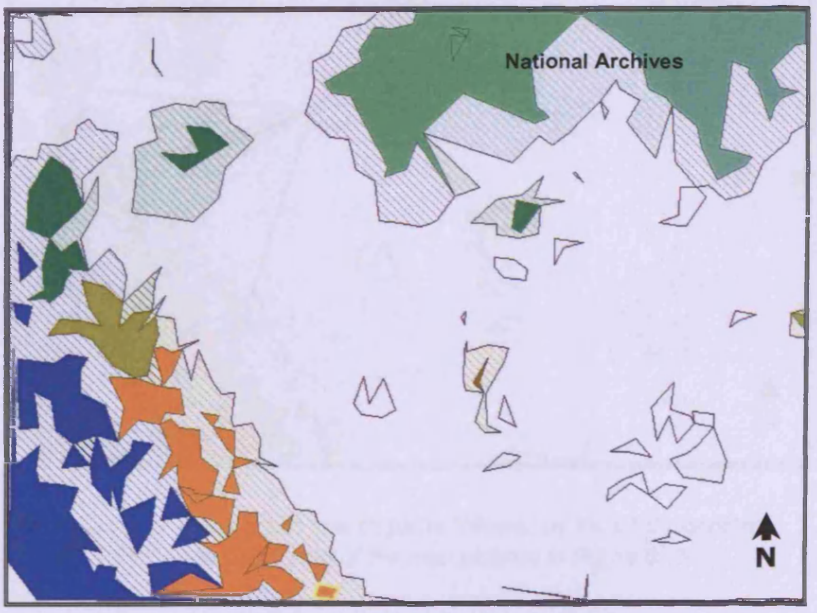


Figure 6.16 – Detail of the southwest part of the map pictured in Figure 6.15.

The reason for the artefacts in both cases is that the enclosing steep regions are comprised of a number of polygons. As the CFS was conceived, these steep polygons in direct contact with the CFS's root are identified in the analysis process to correspond to different potential spatial features, and hence different colours are assigned to them accordingly. When traversing the original graph, what happens is that the algorithm traverses from the root polygon (in white) onto flat polygons (like those mapped in blue, orange, green, etc. in the southwest part of Figure 6.17) through those different steep polygons. That is why the mentioned flat polygons do not have the same parent, and thus "inherit" different colours.

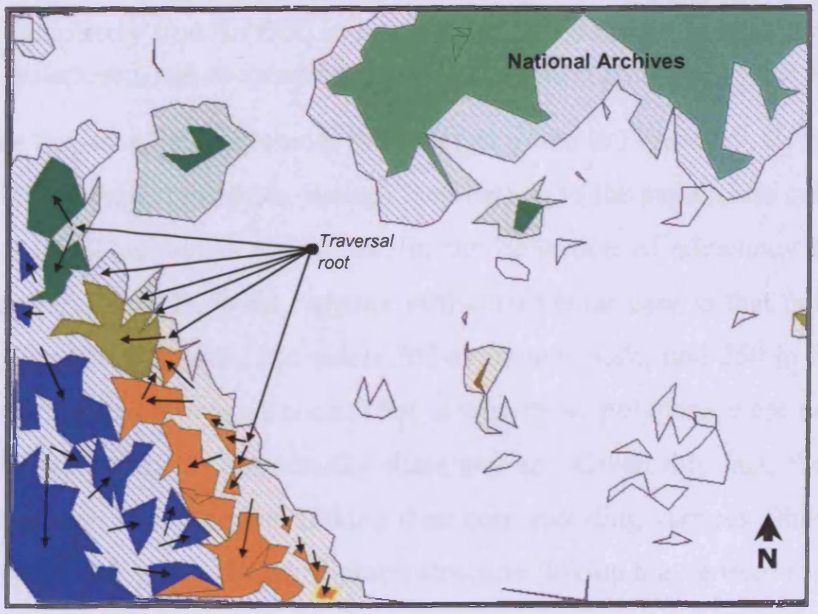


Figure 6.17 – The graph search paths followed by the CFS algorithm in the southwest part of the map pictured in Figure 6.15.

As explained in section 6.3.3, this situation had actually been foreseen. In order to detect and tackle this issue, some analytical rules were also implemented as part of the rationalization procedure (*vd.* Algorithm 6.4). Clearly, given the results, the problem was not effectively resolved. Indeed, the issue was more complex, needed a more consistent solution and thus further investigation was required.

6.3.6 Overcoming limitations

6.3.6.1 The problem

There are two main aspects that explain the unsatisfactory results presented in the previous section:

- Each vertex adjacent to the root in the adjacency graph is assumed to be potentially enclosing a separate containment unit. However, what happens in reality is that some vertices adjacent to the root correspond to a chain of polygons that encloses the same containment unit. Thus, the assumption that each vertex adjacent to the root indicates the potential existence of a different containment unit is wrong.
- Moreover, within a connected component of the graph (typically, a tree traversal's branch), CFS also assumes each relationship of adjacency as being potentially containment. In some cases, however, this is not

completely true. In fact, in some cases two adjacent polygons are simply juxtaposed, not necessarily meaning that one is enclosed by the other.

To illustrate the issue, let us consider the example given in Figure 6.9. Polygons *250*, *256* and *260* are separate entities, though they belong to the same class (*vd.* diagram in Figure 6.18). This fact is mainly due to the definition of adjacency as being a shared edge (section 6.1). What happens in this particular case is that polygon *250* meets polygon *256* at a node, *256* meets *260* at another node, and *260* in turn closes the ring meeting *250* at a third node. That is why these polygons were not merged into one polygon; they do not actually share any arc. Given this fact, there are no edges in the graph of adjacencies linking their corresponding vertices. Shown on top of the polygons in Figure 6.18 is the graph structure linking the vertices representing the polygons.

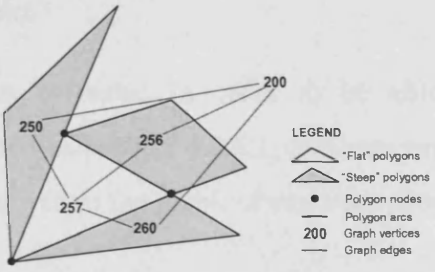


Figure 6.18 - Detail of Figure 6.9: a special case of the spatial relation containment, between a ring of polygons and a single polygon.
(From de Almeida et al.*, in press)*

The example above illustrates a situation that can be detected across the whole map. As noted before, this fact constitutes an issue in the process of the graph analysis, for this case of containment is not explicit in the graph and has to be derived.

Therefore, for an effective interpretation of the urban scene topology, developing CFS simply based upon BFS is not sufficient. The CFS procedure has to be extended in order to be able to detect the spatial relation of containment in a broader sense (*vd.* section 6.1): *polygon-ring containment*. In other words, and as far as the example considered above is concerned, polygons *250*, *256* and *260* have to be considered component parts of a single entity (the ring of polygons), which, for instance, could be represented in the graph of adjacencies by only one vertex, as illustrated by a green ellipse in Figure 6.9.

This may well be achieved by considering the three-polygon ring as a new composite feature comprising three polygonal components. Nevertheless, in order to avoid the creation of new entities (hence the generation of more information), and also in order to preserve the initial structure of the graph of adjacencies, an alternative approach is proposed based on the usage of information already stored in the topological data structure: the spatial relation of *touching* between steep polygons should be taken into consideration. As noted in section 6.1, the spatial relation of *touching* between polygons occurs when two or more polygons do not share an edge but happen to meet at nodes.

Details of the construction of the graph of steep-polygon touchings are given in section 6.2.2.

6.3.6.2 Polygon-ring containment

The CFS procedure was extended in order to be able to detect polygon-ring containments. As explained in section 6.3.6.1, this was implemented by taking into account the information stored in the graph of steep-polygon touchings.

Initially, it was hypothesised that the detection of cycles in the graph of steep-polygon touchings was intrinsically related to the task of detecting polygon-ring containments. However, further examination revealed that by using this method steep-polygon ring containments would not be detectable on the edge of the map. In fact, due to edge effects, chains of steep polygons existing in that area are not closed, and hence do not form a cycle in the associate graph of touchings (*vd.* Figure 6.19).

The possibility of tagging the universe polygon as a "steep" polygon was examined. This would therefore be considered as part of the open chains of steep polygons. However, this process would yield an incorrect result. In fact, as illustrated in Figure 6.19, the universe polygon would form a constituent part of all uncompleted steep-polygon chains lying on the edge of the map, which, in turn, would result in the same single containment unit all around the map edge.

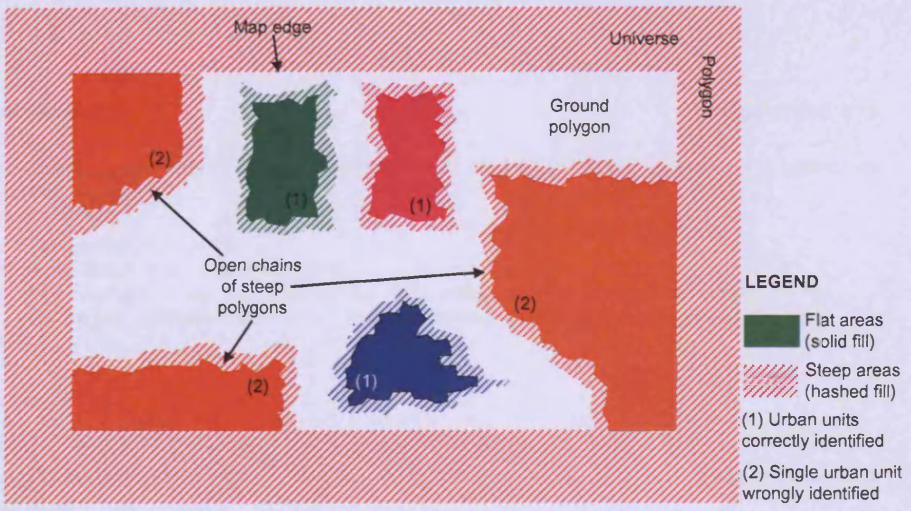


Figure 6.19 – Detection of polygon-ring containments using graph cycles, showing edge effects. (Universe Polygon shown on this figure for illustrative purposes only)

Therefore, although cycle detection may be useful in detecting the spatial relation of polygon-ring containments, this usefulness was not proved due to the afore mentioned edge effects. The general algorithm for graph cycle detection was added to the code, but no further implementation to investigate the use of graph cycles for detecting polygon-ring containments was attempted. These issues were resolved as described below.

The original graph of adjacencies is traversed. When visiting the adjacent steep vertices of the root, the CFS algorithm takes the first vertex appearing in the root's linked list and, starting from this one, traverses the graph of steep-polygon touchings. Because the graph of touchings is a disconnected graph, the traversal process covers only the subgraph that the given steep vertex belongs to. While traversing this particular subgraph, the CFS algorithm tags all the steep vertices visited as belonging to the same connected unit. This process continues until the first level of adjacency of the graph of adjacencies is exhausted. When the CFS comes across a root's adjacent vertex already tagged as belonging to a particular containment unit, this is skipped and the corresponding polygon remains intact, belonging to the containment unit already identified.

PolyRing_containment Algorithm 6.5

```
Graph of steep-polygon touchings is traversed starting from vertex v (based on
depth-first search):
Vertex v is visited;
For vertex t adjacent to v Do
    If t hasn't been visited yet Then
        Tag vertex t as belonging to the same polygon-ring as that of v;
        PolyRing_containment(t); (take vertex t and proceed recursively)
    End if
End for
End
```

Containment_first_search Algorithm 6.6

```
Graph of adjacencies is traversed (based on breadth-first search):
Start counting the nbr of containment units;
While Queue of vertices to be visited is not empty Do
    Visit vertex k;
    Start counting the valence of k;
    For each adjacent vertex v to k Do
        If k ≠ root Then
            Count the valence of k;
        End if
        If v hasn't been visited yet Then
            If k is the root Then
                v's parent = root;
                v's level of containment = 1;
                If v hasn't been tagged yet as belonging to a certain
                containment unit Then a new polygon-ring containment
                enclosing a new containment unit was detected:
                    Count nbr of containment units;
                    v's containment unit = nbr of units counted so far;
                    PolyRing_containment(v); //all steep polygons t touching
                    v are tagged as belonging to the same polygon-ring as
                    that of v -> Algorithm 6.5
                End if
            Else
                v's parent = k;
                v's containment unit = k's containment unit;
                v's level of containment = level of containment of v's
                    parent+1;
                //v's level of adjacency represents a level of containment
                within the current containment unit
            End if
            Put v in the Queue to be visited thereafter;
        End if
    End for
Root's "valence" = nbr of containment units (recall that steep-polygon islands are
discarded);
End while
Visual representation of the graph analysis results;
End
```

To illustrate the concept implemented, let us take a simpler scene pictured in Figure 6.20. Let us suppose that steep polygons *3,...,11* (in dark green) are constituent parts of the rings of steep polygons enclosing flat polygons *12* and *13* (*vd.* Figure 6.20a); in other words, there is a sub-graph of the graph of steep-polygon touchings which consists of vertices *3* to *11*.

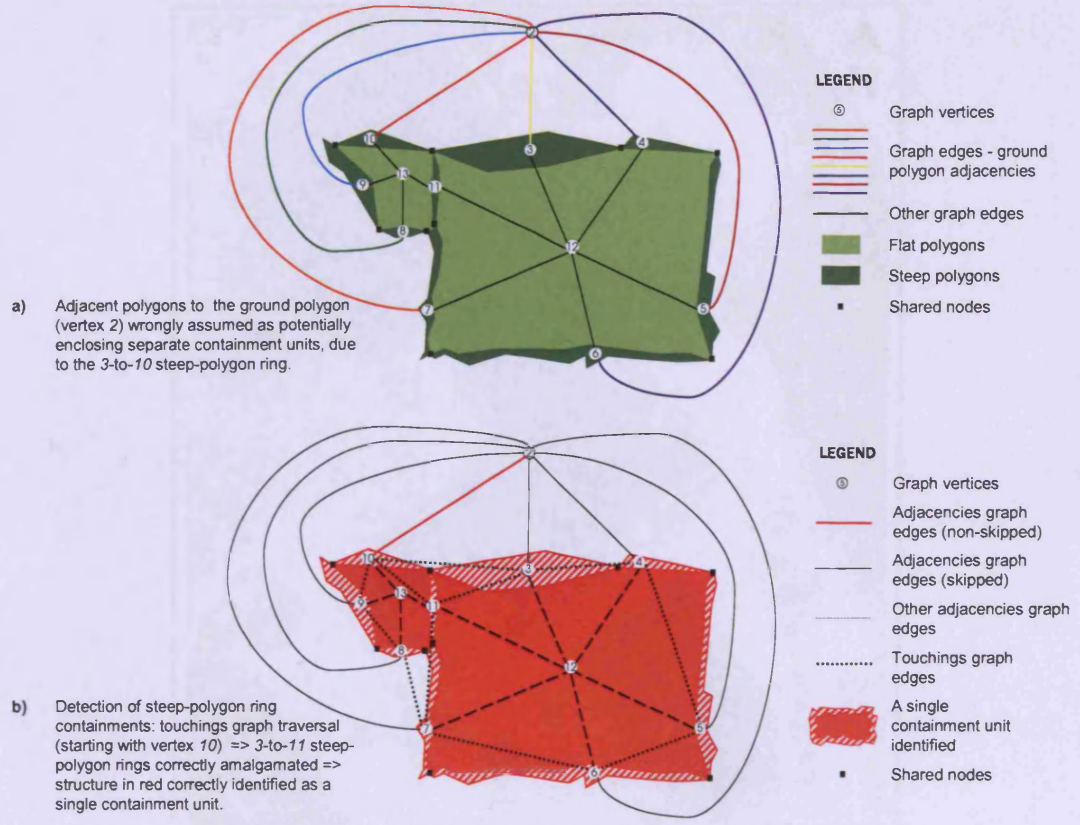


Figure 6.20 - The containment-first search process:
a) before polygon-ring containments were detected; b) after polygon-ring containments were detected.

When vertex *2* is visited in the adjacencies graph, the algorithm takes the vertex at the top of *2*'s adjacency list, vertex *10*, and the graph of steep-polygon touchings is traversed starting from *10*; all the other steep vertices belonging to the same sub-graph as that of *10* are tagged accordingly, indicating a potential containment unit. When vertex *10* is exhausted in the process of traversing the adjacencies graph, CFS moves on to visit vertex *9*; this is now skipped since it was previously tagged as belonging to an identified containment unit. And so on so forth until vertex *3* is visited, and the containment unit is complete. Visually, the translation of the facts above is accomplished by assigning the same colour to all steep (hashed pattern) and flat (solid colour) polygons within the same containment unit (*vd.* Figure 6.20b).

The graph analysis program, with the improved CFS, was applied again to the case study of the London (Kew) dataset. The results obtained are depicted in Figure 6.21.

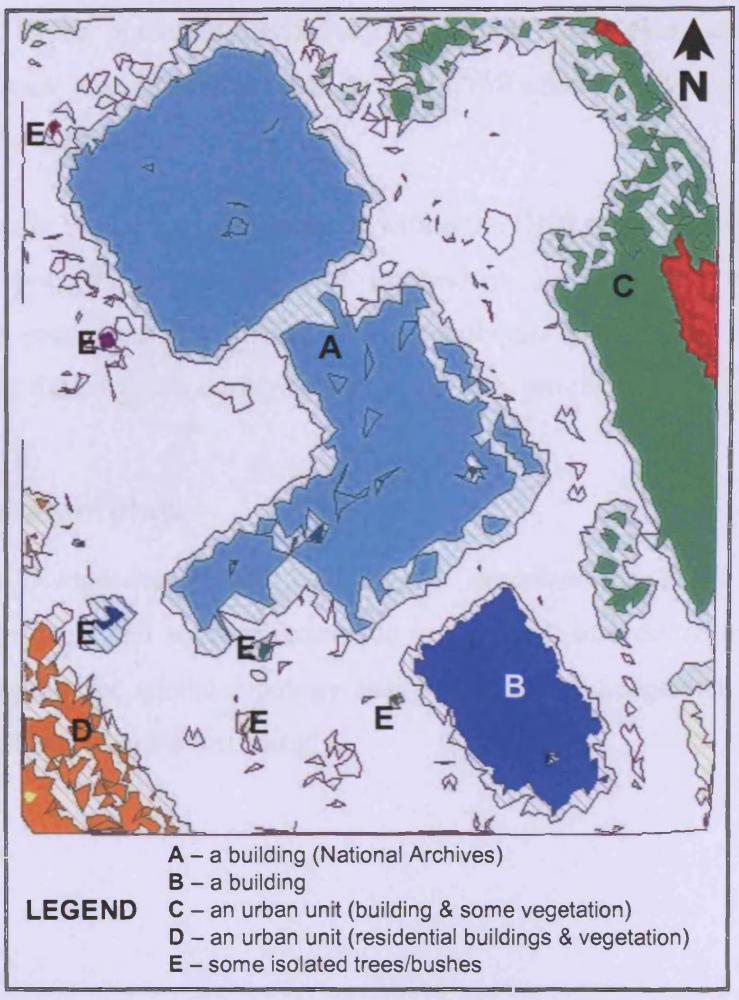


Figure 6.21 – A second experiment carried out with the improved CFS, extended to detect polygon-ring containments. Experiment applied to the case study area of the London dataset. (The letters on the map are polygon labels in insert)

In visual terms, the potential existence of spatial features is much clearer now: different colours represent disjoint containment units - solid colours correspond to flat polygons, and coloured hashed patterns correspond to steep polygons; the higher the level of adjacency/containment of polygons within a given unit, the darker the tone of the colour assigned to it.

It should be noted that the steep polygons (and enclosed flat polygons) mapped in red in the northeast part of the case study area were wrongly tagged, and hence assigned the wrong colour. They constitute upper levels of adjacency within the containment unit mapped in green, and should have been mapped with a darker tone of that colour. This fact was due to edge effects. In fact, during the adjacency graph search, the search process happened to traverse from the UEB to the dark red polygon via the universe polygon; the universe polygon was therefore assumed by the algorithm as

"steep", and to be potentially enclosing a separate unit; thus, an independent containment unit was identified, and hence a different colour from green was assigned by the algorithm.

As can be seen in Figure 6.21, for instance within the UEB mapped in white, the arcs of the steep-polygon islands are still mapped to illustrate the extent of the generalisation procedure. Recall that all steep-polygon islands are "merged" with their enclosing flat polygons during the rationalization process.

6.3.7 Coding the algorithm

In his books, Sedgewick (1988, 1998, 2002) describes the main steps of the algorithms presented and some pseudo-code is provided. In order to implement the computer program for spatial topology analysis, some of Sedgewick's algorithms were borrowed and adapted accordingly.

6.3.7.1 The main program

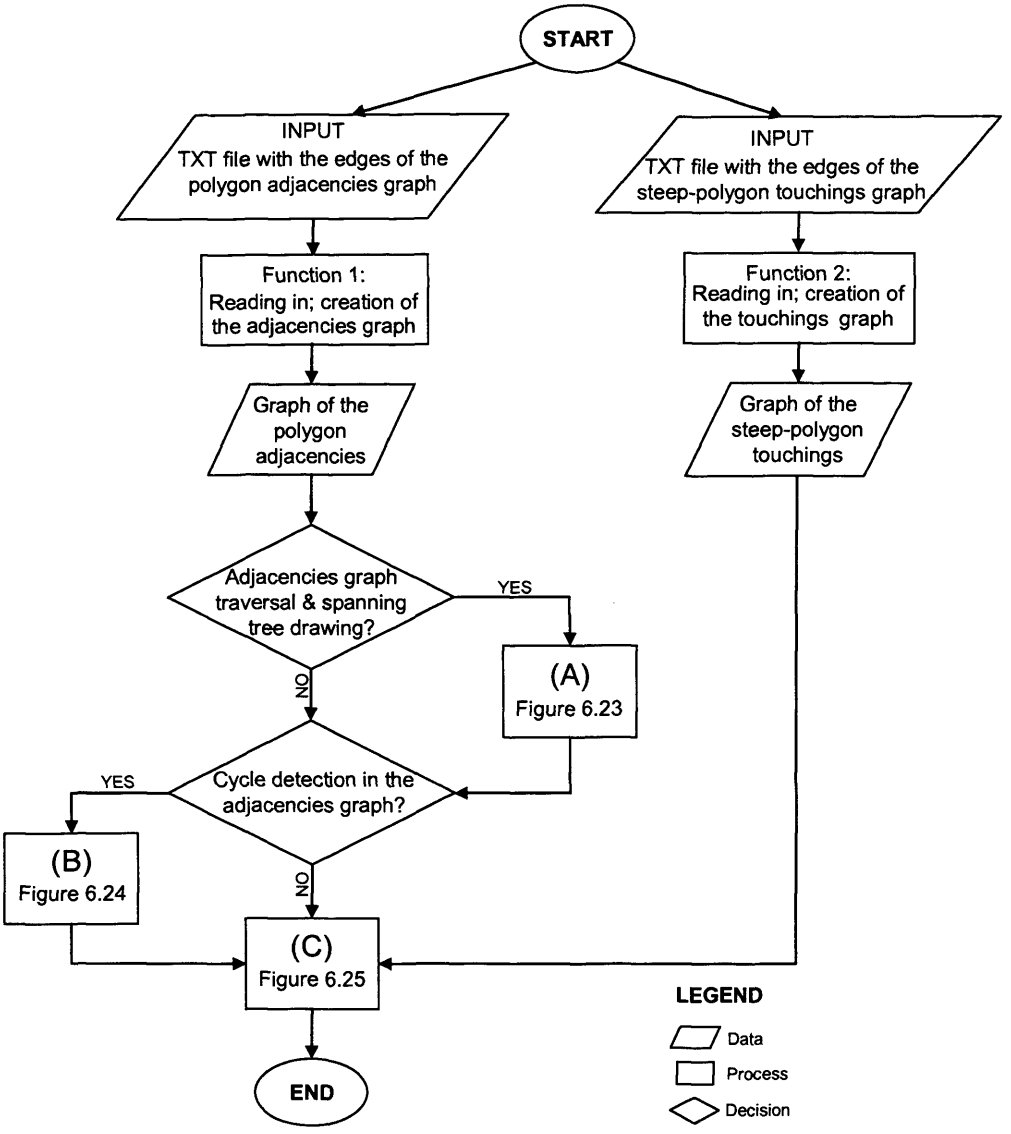


Figure 6.22– Overview of the spatial topology analysis program.

The flowchart depicted in Figure 6.22 gives an overview of the program implemented in C for the analysis of spatial topology, and shows the flow of the data and the processes undertaken. The functions which implement the various processes are indicated in the flowcharts, which also show how the different functions are interconnected to build up the whole algorithm. Functions are numbered and their associate code can be found in Appendix E. The elements (A), (B) and (C) are decomposed in Figure 6.23, Figure 6.24, and Figure 6.25 respectively.

In terms of pseudo code, the main program can be summarised as follows.

Main Algorithm 6.7

```
// "What's the total number of polygonal regions?"
    Read(nbrpolys);
// "Graph traversal using: DFS or BFS algorithm?"
    Read(Option 2);
// "Spanning-tree output file as a: Text Data File or Commented File?"
    Read(Option 1);
//Dynamic allocation of memory & creation of the array of linked lists adj[]:
    INIT_adj (vd. Algorithm 6.1);
//Dynamic allocation of memory & creation of the array of linked lists touch[]:
    INIT_touch (vd. Algorithm 6.2);
//Dynamic allocation of memory & initialisation of array:
For i Do
    visited[i] = -1;
End for
If nbr of edges of graph adj[] ≥ nbr of vertices of adj[] Then graph of
        adjacencies adj[i] has cycles:
        // "Graph cycles detection?"
        Read(Option 3);
End if
//Search the graph of adjacencies adj[] nbrpolys times, starting from a
different vertex i at a time:
For each different vertex i Do
   If Option 2 ≠ "No" Then generate the resulting spanning-tree:
        //Create the output file to write the spanning-tree edges to:
        Create_spantree_file(outputtree_i.dat); (vd. Algorithm 6.8)
        //Open created output file;
        If Option 1 = "Commented File" Then
            //Prepare "comments" to be written in the output file
        End if
        For i Do
            visited[i] = 0; //none of the vertices of the adjacencies graph
                            has been visited
        End for
        //Traverse the graph adj[], starting from vertex i:
        If Option 2 = "DFS algorithm" Then
            Depth_first_search(i); (vd. Algorithm 6.9)

        Else
            Breadth_first_search(i); (vd. Algorithm 6.10)
        End if
   End if
   If Option 3 = "Yes" Then
        //Create/open the output file:
        Create_cycles_file(cycles.dat); (vd. Algorithm 6.8)
        CycleDetection_DFS(i); (vd. Algorithm 6.11)
   End if
//Create/open the output file to receive the CFS tree edges and levels of
containment:
   Create_CFStree_file(CFStree_i.dat); (vd. Algorithm 6.8)
//Initialisation of array:
        polyclr[i] = -9; //to store the attributes of the vertices of adj[]
//Detection of different levels of containment / Identification of containment
units:
   Containment_first_search(i); (vd. Algorithm 6.12)
//Create/open the output file to write in the final results of the urban
topology analysis - CFS and Rationalization procedures:
   Create_CFS&Rationalization_file(CFS&Rationalization_i.dat);
   (vd. Algorithm 6.8)
   Rationalization(i); (vd. Algorithm 6.15)
End for
End
```

6.3.7.2 Dissection of process (A): adjacencies graph traversal

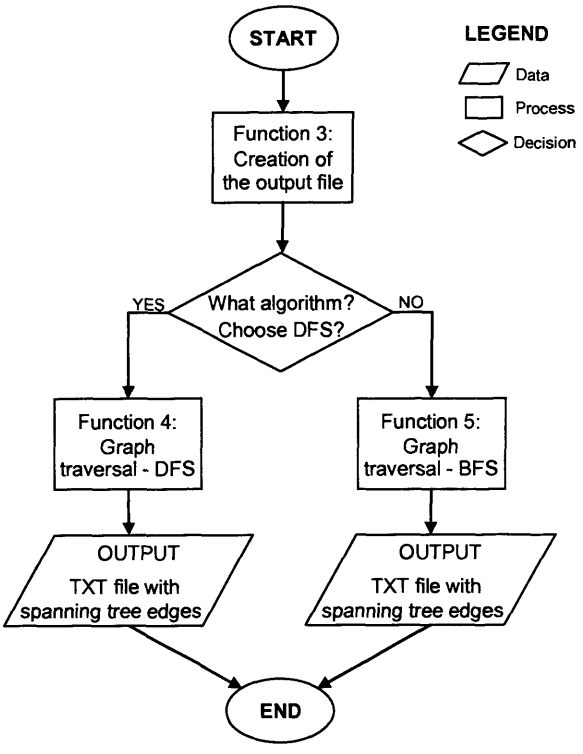


Figure 6.23 – The process of traversing the graph of adjacencies by using either the DFS or BFS algorithm; generation of the respective spanning tree. (Process (A) in the flowchart depicted in Figure 6.22)

After being loaded, the graph of polygon adjacencies is stored in the array of linked lists adj[] (Function 1 in Figure 6.22). The user can choose then whether to generate a graph spanning tree or not. If so (process (A) in Figure 6.22), two algorithms are available for the purpose; indeed, as noted before, for the sake of flexibility two different algorithms to traverse a graph were implemented: the DFS (*vd.* Algorithm 6.9) and BFS (*vd.* Algorithm 6.10). Functions 4 and 5 implement respectively these two algorithms, and can alternatively be chosen by the user (*vd.* Figure 6.23).

The main aim of the process illustrated in Figure 6.23 is the generation of the adjacency graph spanning trees. Again for the sake of flexibility, all possible spanning trees are created. In other words, supposing that the adjacency graph comprises N vertices, then it is traversed N times, starting from each of its vertices, and the respective spanning tree is generated. Their edges are written into the output text files *outputtree_#.dat*. This output file is created by Function 3 (*vd.* Figure 6.23), whose implementation is based on Algorithm 6.8. As an example, there is a printout

in Appendix F of the output file *outputtree_003.dat* for the London case study (graph search starting from vertex *3*).

Create_output_file Algorithm 6.8

```
//Define file name head (the path):
    filename_head = …;
//Define file name tail:
    filename_tail = ".dat";
//Allocation of memory for the number of the file & initialisation:
    filename_num = malloc(nbr of graph vertices);
//Allocation of memory & initialisation of the file final name, totalfinalname;
//Composition of the final string for totalfinalname;
//Return totalfinalname;
End
//OUTPUT: the output file name.
```

Depth_first_search Algorithm 6.9 (Adapted from Sedgewick, 1998, 2002)

```
//INPUT: a graph
//Take the input graph
//Traverse adj[] starting from a given vertex i:
    //Vertex i is visited:
        visited[i] = 1;
    //Count level of depth;
    For all vertices t adjacent to i Do
        If t not visited yet Then
            //Write in the output file, outputtree_#.dat, the new tree edge:
              i-t;
            Depth_first_search (t); //proceed recursively with t
        End if
    End for
End
//OUTPUT: DFS spanning tree of the initial graph
```

Breadth_first_search Algorithm 6.10 (Adapted from Sedgewick, 1998, 2002)

```
//INPUT: a graph
//Take the input graph
//Initialise Queue of the graph vertices to be visited:
    QUEUEinit(total nbr of graph vertices);
//Dynamic allocation of memory & initialisation of the array:
For i Do
    marked[i] = 0; //indicates which graph vertices are already in the Queue to be
                     visited
End for;
//Traverse adj[] starting from a given vertex i:
    //i is put in the Queue:
        QUEUEput(i);
        marked[i] = 1;

    While Queue of vertices to be visited is not empty Do
        //Take the front vertex in the queue, k
        //Vertex k is visited:
            visited[k] = 1;
        //Count levels of adjacency;

        For all vertices t adjacent to k Do
            If t hasn't been visited yet Then
                //Write in the output file, outputtree_#.dat, the new tree edge:
                k-t;
                //t is put in the Queue to be visited thereafter:
                QUEUEput(t);
                marked[t] = 1;
            End if
        End for
    End while
End
```

```
//OUTPUT: BFS spanning tree of the initial graph
```

6.3.7.3 Dissection of process (B): adjacency graph cycles detection

Another optional task of the main program (process (B) in Figure 6.22) is the detection of cycles in the original graph of adjacencies. As explained in the previous chapter, the general algorithm for cycle detection was initially implemented because it was believed that cycles of the adjacency graph were intrinsically related to the spatial relation of polygon-ring containment. As explained in section 6.3.6.2, the detection of this spatial relationship was eventually addressed in a different way, and hence cycle detection was not used in the analysis of the urban topology. However, the cycle detection associate code was left in place to support possible further investigation of this approach.

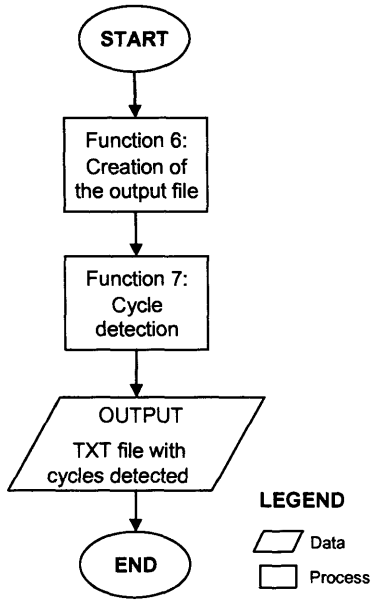


Figure 6.24 – Detection of cycles in the graph of adjacencies. (Process (B) in the flowchart depicted in Figure 6.22)

Function 7 in Figure 6.24 implements the cycle detection procedure which is based upon the DFS algorithm: when visiting a particular vertex during the traversal process, DFS finds a cycle every time an edge leading to a vertex already visited is detected. This procedure can be summarised in terms of pseudo code as described below in Algorithm 6.11.

```
CycleDetection_DFS          Algorithm 6.11 (Adapted from Sedgewick, 1998, 2002)

//INPUT: a graph
//Take the input graph
//Traverse adj[] starting from a given vertex i:
    //Vertex i is visited:
        visited[i] = 1;
    //Count level of depth;
    For all vertices t adjacent to i Do
        If t not visited yet Then
            CycleDetection_DFS(t); //proceed recursively with t
        Else
            If t = i's parent Then
            //Write in the output file, cycles.dat ("Pointing back to parent")
            Else
            //Write in the output file, cycles.dat ("Cycle detected: i-t)
            End if
        End if
    End for
End
//OUTPUT: DFS spanning tree of the initial graph with all cycles indicated
```

The cycles detected in an adjacency graph are written in the output text files *cycles.dat*. This output file is created by Function 6 (*vd.* Figure 6.24) whose implementation is based on Algorithm 6.8. As an example, there is a printout in Appendix G of the output file *cycles.dat* for the London case study area (graph search starting from vertex *3*).

6.3.7.4 Dissection of process (C): containment-first search and rationalization

The main contribution of the author to the spatial topology analysis program refers to process (C) in the algorithm (*vd.* Figure 6.22 and Figure 6.25).

As explained in section 6.2.3.2, at the very beginning of the execution of the program, Function 2 (*vd.* Figure 6.25) reads in the steep-polygon touching graph edges, from the input file *outputnodeshare#.dat*, and stores them in the array of linked lists touch[] (*vd.* Algorithm 6.2).

In turn, both Functions 8 and 9 (*vd.* Figure 6.25) implement Algorithm 6.8 to create the output files *CFStree_#.dat* and *CFS&Rationalization_#dat*, respectively. *CFStree_#.dat* are commented files with the spanning tree originated by the CFS algorithm, indicating all the levels of containment found during the traversal process (starting from vertex #). *CFS&Rationalization_#dat* files store the results of the urban topology analysis, (traversal procedure starting from vertex #).

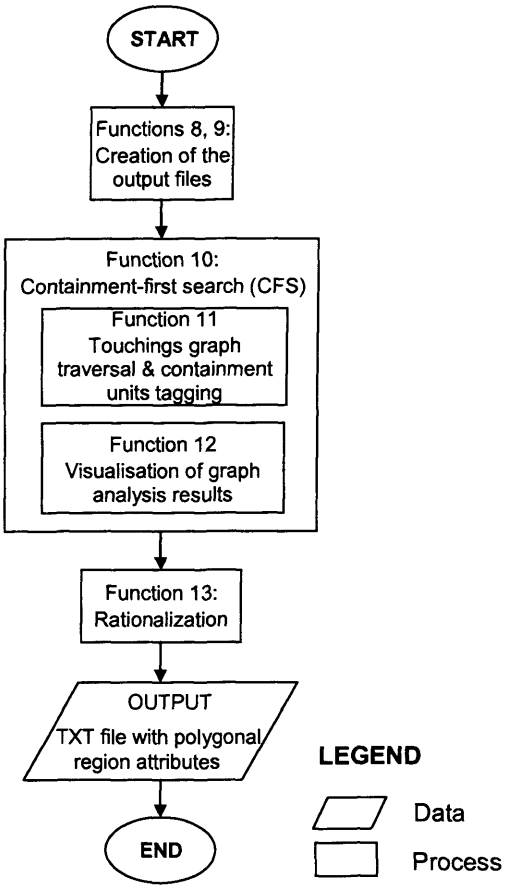


Figure 6.25 – The CFS & Rationalization process.
(Process (C) in the flowchart depicted in Figure 6.22)

After creating and opening the output files, Function 10 (*vd.* Figure 6.25), which implements the containment-first search procedure, is called (*vd.* Algorithm 6.12). This procedure investigates where the spatial relation of polygon-ring containment occurs between steep polygons, and identifies the containment units as it goes (Function 11 in Figure 6.25; *vd.* Algorithm 6.13). The results of the analysis process carried out by Function 10, along with Function 11, are translated by Function 12 (*vd.* Figure 6.25) into something that can be visualised. As described in section 6.3.4, a gamut of different colours, a gradient of their tones, and different filling styles were used to map the different potential urban units detected, and at the same time to show different levels of adjacency/containment within each urban unit. Finally, Function 13 (*vd.* Figure 6.25) clears out all steep polygon islands within flat regions, and writes the new attributes of the slope regions in the output text files, *CFS&Rationalization_#.dat*: the polygonal region ID; the colour components, Hue, Saturation, and Value; a colour unique value; and the colour filling style (0 – solid, for flat regions and cleared-out steep polygons; 1 – hashed pattern, for the remaining

steep regions). As an example, there is a printout in Appendix H of the output file *CFS&Rationalization_003.dat* for the London case study (graph search starting from vertex *3*).

Containment_first_search Algorithm 6.12

```
//INPUT: graphs adj[] and touch[]
//Graph of adjacencies adj[] is traversed (based on breadth-first search),
  starting from a given vertex k:
    root = k;
//Dynamic allocation of memory & initialisation of arrays:
For all vertex i of adj[] Do
    marked[i] = 0; //indicates whether vertex i of the adjacencies graph is
                     already in the Queue, to be visited
End for;
For all vertex j of touch[] Do
    checked[j] = 0; //indicates whether vertex j of the touchings graph has
                      already been visited
End for;
//All adjacencies graph vertices i are tagged as "not visited":
For all vertex i Do
    visited[i] = 0;
End for;
//Level of containment of the traversal root is set to "0":
    polyclr[root].containment_level = 0;
//Containment unit of the traversal root is set to "0":
    polyclr[root].Containment _unit = 0;
//Start counting number of containment units:
    countUrbanSets = 0;
//Initialise Queue of the graph vertices to be visited:
    QUEUEinit(total nbr of graph vertices);
//First graph vertex to be put in the Queue is the traversal root:
    QUEUEput(root);

While Queue of vertices to be visited is not empty Do
    //Take the front vertex in the queue, k
    //Vertex k is visited:
          visited[k] = 1;
    //Start counting the valence of k:
        countValency = 0;

    For each vertex v adjacent to k Do
        If k ≠ root Then count the valence of k:
                countValency++;
        End if
        If v hasn't been visited yet Then
            If k is the root Then
                v's parent = root;
                v's level of containment = 1;
                If v hasn't been tagged yet as belonging to a containment unit
                Then new polygon-ring containment, potentially enclosing a
                spatial unit, was detected:
                    countContainmentUnits++;
                    v's containment unit = countUrbanSets;
                    PolyRing_containment(v); //all steep polygons t touching
                                               v are tagged as belonging to
                                               the same polygon-ring as that
                                               of v - vd. Algorithm 6.13)
                End if
            Else
                  v's parent = k;
                  v's containment unit = k's containment unit;
                  v's level of containment = level of containment of v's
                         parent+1 //v's level of adjacency represents a level
                         of containment within the current containment unit);
            End if
            //Put v in the Queue to be visited thereafter:
                QUEUEput(v);
                marked[v] = 1;
        End if
    End for
    //k's valence is the nbr of its adjacent vertices v counted above:
```

```
        k's valence = countValency;
End while
//Root's "valence" is the nbr of containment units (steep-polygon islands were
discarded):
    root's valence = countContainmentUnits
Colouring; (vd. Algorithm 6.14)
End
//OUTPUT: the containment units and their attributes
```

PolyRing_containment Algorithm 6.13

```
//INPUT: touch[] graph; a given graph vertex v
//Graph of steep-polygon touchings is traversed (based on depth-first search),
  starting from vertex v:
//Vertex v is visited:
    checked[v] = 1;
For all vertices t adjacent to v Do
    If t hasn't been visited yet Then
        t's containment unit = countContainmentUnits;
        //Tag vertex t as belonging to the same containment unit as that of v;
        //Vertex t also belongs to the same polygon-ring as that of v;
        PolyRing_containment(t); //take vertex t and proceed recursively
    End if
End for
End
//OUTPUT: the potential containment units detected
```

Colouring Algorithm 6.14

```
//INPUT: adjacencies graph adj[]
//Dynamic allocation of memory & initialisation of the array
For all containment units i Do
    nbrLevels_Unit[i]; //to store the number of levels of containment of the
                          containment unit i
End for;
//Calculate the maximum number of levels of containment per containment unit;
//Computation of dH (increment to calculate a different colour for each
  containment unit):
    dH = (int) (floor(360 / nbr of containment units));
//Dynamic allocation of memory for the array
dSV[i]; //increment to calculate different tones of the colour assigned to the
        containment unit i
//Computation of dSV for all containment units i:
    dSV[i] = (int)(floor(130 / nbrLevels_ Unit[i]));

For each vertex i of the graph of adjacencies adj[i] Do
    If i is at an "even" level of containment //i.e. associate polygon is a
       flat Then Polygon fill style = 1 //solid colour
    Else
        Polygon fill style = 0 //hashed pattern colour
    End if
    If i is the root vertex Then
        //Associate polygon is mapped in white (H = 60, S = 0, V = 100)
    Else
        i's H = i's containment unit * dH;
        i's S = dSV[i] * i's level of containment;
        If i's S ≤ 100 Then
            i's V = 100;
        Else
            i's V = 200 - i's S;
            S = 100;
        End if
    End if
End for
End
//OUTPUT: the containment-unit attributes
```

Rationalization Algorithm 6.15

```
//INPUT: adjacencies graph adj[]
For each vertex i of adj[] Do
```

```
If i is single-valent AND represents a steep polygon Then i becomes one with
its enclosing polygon:
          i's H = H of i's parent;
          i's S = S of i's parent;
          i's V = V of i's parent;
    End if
//Write in the output file i's attributes: ID, colour components (H, S, and V),
  colour code, colour fill style;

End for
End
//OUTPUT: txt file with the containment-unit attributes
```

Given the fact that the problem of detecting the steep-polygon containments was resolved by the improved CFS algorithm, the rules initially implemented in Algorithm 6.2 in order to address that issue turned out to be redundant. Thus, as can be seen above, Algorithm 6.15 does not include those rules; however, the associate code was left in place and is still operating.

6.4 Visualisation of the analyses

The human brain is sensitive to the visual representation of real scenes, and visual analysis can often reveal patterns not discernable by current automated analysis techniques. Thus, the incorporation in the application of capabilities for the spatial representation of the urban topology appeared to be relevant. Given the dimension and complexity of the original graphs of adjacencies, it is believed that the observation of the traversal trees is useful in detecting the existence of urban features.

In addition, the possibility of linking up the graph analysis application with the GIS environment was also investigated. If the visualisation tool is coupled with the original map of steep/flat regions, the utility of the spatial representation of the urban topology should be enhanced. The ultimate goal was the implementation of functionalities for dynamic display of that map according to the results of the urban topology analysis.

For this purpose, an interactive tool was developed. This was implemented in ArcMap (ArcGIS 8.3) using its embedded programming language, Visual Basic for Applications (VBA). The design of this tool and its implementation are described below in sections 6.4.1 and 6.4.2.

6.4.1 Designing the interactive tool

The key aspect in the implementation of this application was the use of the ArcGIS object library, called *ArcObjects*. ArcObjects is a collection of objects behind the menus and icons used to perform tasks in ArcGIS. These same objects also enable programmers to access data and to perform tasks programmatically; indeed, ArcInfo functionalities can be customised and extended at the lowest level, or new GIS applications can be constructed (Chang, 2005). In this research, ArcObjects were used to customise a series of functionalities in ArcMap in order to build up the prototype of an interactive tool for the visualisation of spatial topology.

The prototype implemented in VBA, utilising ArcObjects, comprises a VB *module* and three *user forms* (*vd.* printout of these components and associate code in Appendix I). The diagram depicted in Figure 6.26 gives an overview of the application developed, and shows how the implemented functionalities are linked together.

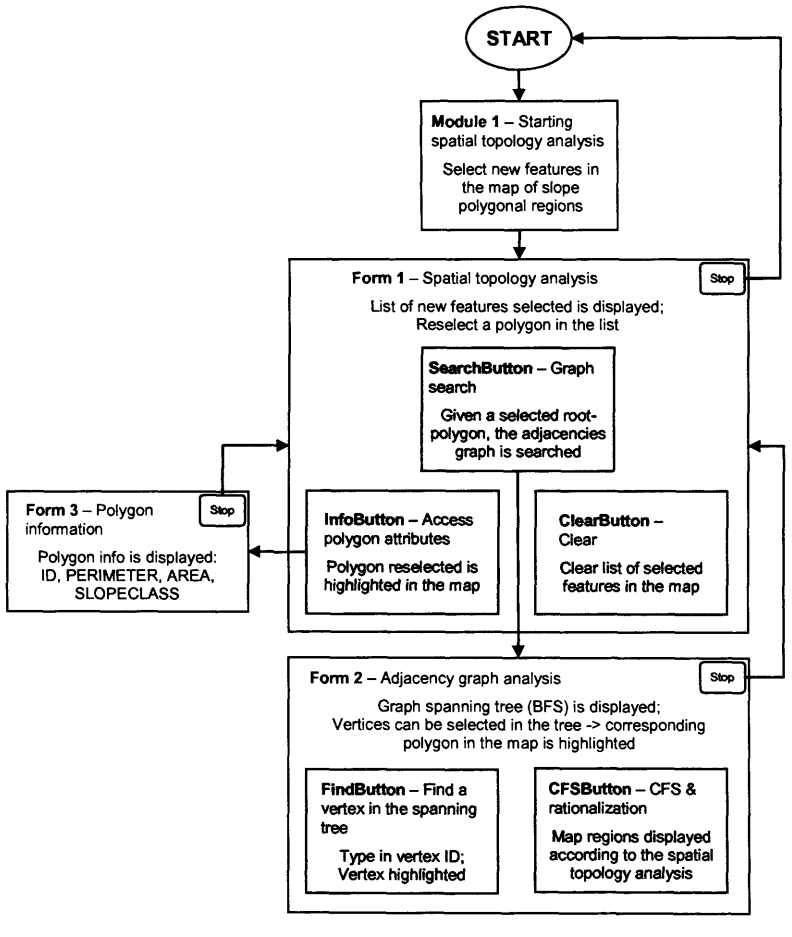


Figure 6.26 – A prototype of an interactive application for the visualisation of spatial topology.

6.4.2 Coding the interactive tool

Once the application starts running, the user is asked to select one or more polygons on the steep/flat region map (*e.g.* the polygon(s) in an area that the user may be interested to investigate) (*vd.* Figure 6.27).

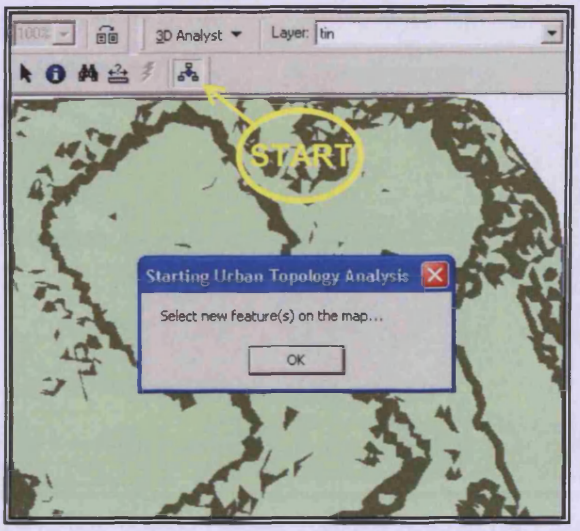


Figure 6.27 – Starting the interactive application: selection of features on the steep/flat region map.

After new feature(s) are selected on the map, a window comes up (Form 1 object, Figure 6.26) where the IDs of the selected polygons are listed (*vd.* Figure 6.28). By choosing from the selected polygon list one polygon at a time, the user is able to quickly access some of its attributes (InfoButton, Figure 6.26; "Polygon info" button, Figure 6.28): ID, perimeter, area, and gradient class. A new window with these attributes pops up on the screen, and the respective polygon is highlighted on the map (*vd.* Figure 6.28).

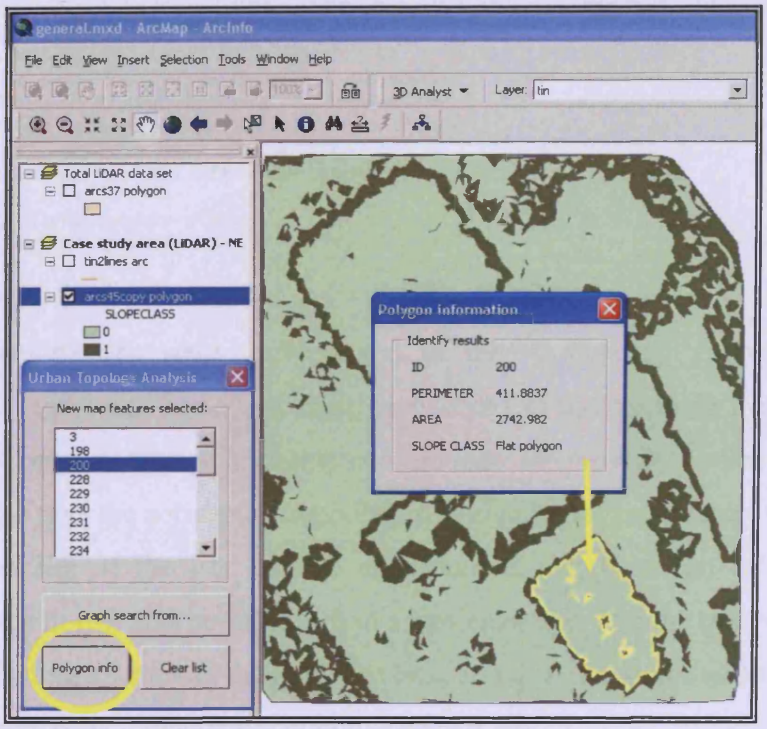


Figure 6.28 – Form 1 object - Spatial Topology Analysis (vd. Figure 6.26); accessing the selected polygon's information (highlighted in yellow on the map). (The London case study)

Examining the original map of steep/flat polygons, the user can carry out the traversal of the graph of polygon adjacencies, starting from any vertex. To traverse the graph, the user presses the SearchButton (Figure 6.26; "Graph search from…" button, Figure 6.28), after setting the root polygon for the tree. Setting the root can be done by selecting a polygon from the list displayed. Accordingly, the application takes the respective output file from the C program, *outputtree*_root.*dat*, and the spanning tree edges are read in.

Once the spanning tree is uploaded, it is constructed and displayed in a different window using the Windows TreeView control (Form 2 object, Figure 6.26; "Adjacency Graph Analysis" window, Figure 6.29). The generation and display procedure of the uploaded spanning tree is summarised below in terms of pseudo code (*vd.* Algorithm 6.16).

Private Sub UserForm2_Activate() Algorithm 6.16

```
The user chooses on Form 1 object the search root (one from the listed polygons);
Check out the ID# of the selected root;
Set the input file (outputtree_root.dat) with the spanning tree to be uploaded:
    File name_num = root's ID;
    File name_head = location of the file (path);
    File name_tail = ".dat";
```

```
    Input file name = (name_head & name_num & name_tail);
Initialisation of the SpanningTree (Windows TreeView control):
    Add the first tree vertex: root;
While not end of the input file Do
    Read in each line at a time (a tree edge):
        Read in first vertex (left-hand side);
        Read in second vertex (right-hand side);
        Add vertices read to the tree;
        Display the new tree edge expanded;
End while
Set SpanningTree border style;
Set SpanningTree indentation;
End
```

Several functionalities were implemented in the "Adjacency Graph Analysis" window (*vd.* Figure 6.29) to allow visual inspections of the spanning tree displayed. Active links between tree components and the map are provided: when a vertex is selected in the tree, the corresponding polygon is highlighted on the map (in yellow); relevant branches of the tree can be expanded; if not, they can be hidden by contracting them; it is also possible to find a particular vertex in the tree by typing its ID in the "Finding a vertex in the tree" text box, and then pressing FindButton (Form 2 object, Figure 6.26; "Search>>" button, Figure 6.29).

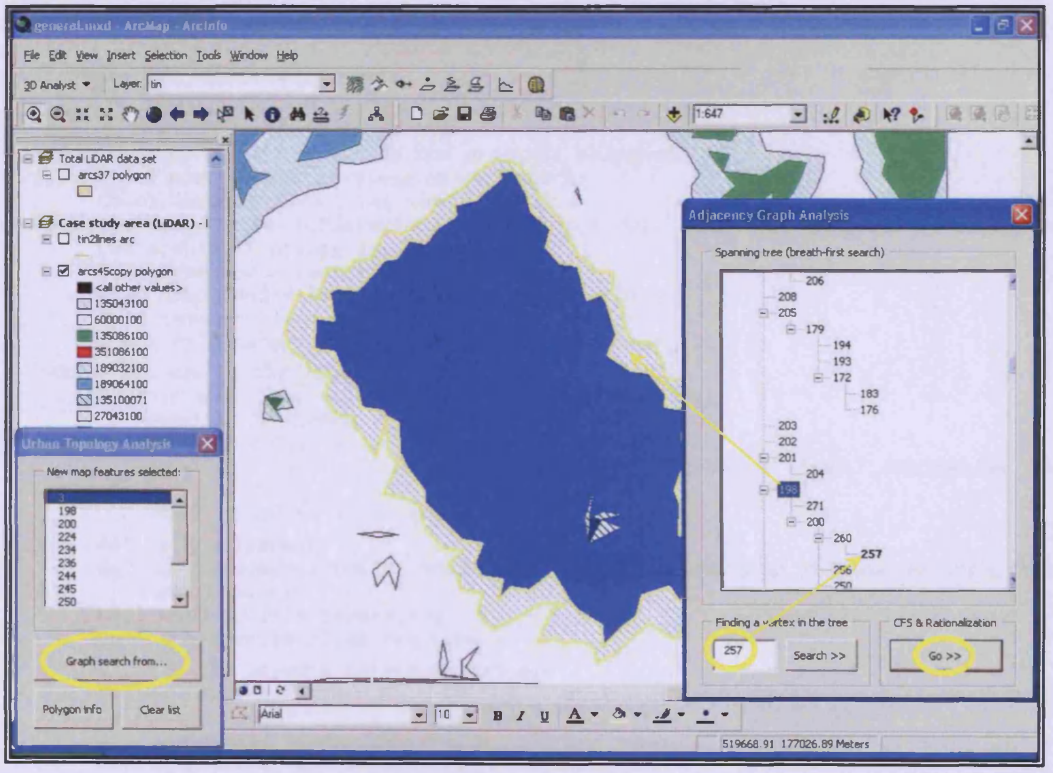


Figure 6.29 – Construction and display of the spanning tree of the adjacency graph; dynamic mapping of the slope regions in the original map according to the CFS and Rationalization results. (The London case study)

Finally, the results of the containment-first search and rationalization procedures can be visualised when CFSButton is pressed (Form 2 object, Figure 6.26; "Go>>" button, Figure 6.29). The application uploads the results obtained when the graph of adjacencies is searched from the previously chosen root-vertex. This is accomplished by reading in the respective output file from the C program, *CFS&Rationalization_*root*.dat*. The attributes read in are used to populate the fields HUE, SATURATION, VALUE, FILLSTYLE, and COLOUR of the PAT info file (*vd.* Figure 6.30); a colour gamut is generated based on the unique values of the field COLOUR. The steep/flat regions in the original map are then redisplayed according to the new attributes mentioned above: the polygons tagged as constituent parts of the same containment unit are mapped in different tones of the same colour; the different tones indicate different levels of containment within the specified unit; and hashed patterns indicate rings of steep regions enclosing flat regions (in solid colour).

The process described above is summarised in Algorithm 6.17 in terms of pseudo code.

Private Sub CFSButton_Click() Algorithm 6.17

```
Set up the map document in ArcMap;
Find the map layer containing the gradient polygons;
Given the root of the previous graph search:
    Check out the ID# of the root;
Set the input file (CFS&rationalization_root.dat) with the respective results of
    the spatial topology analysis:
    File name_num = root's ID#;
    File name_head = location of the file (path);
    File name_tail = ".dat";
    Input file name = (name_head & name_num & name_tail);
While not end of the input file Do
    Read in each line at a time (a polygon attributes):
        Read in ID number (id#);
        Read in colour's unique value (colour);
        Read in colour components Hue-Saturation-Value (hue, saturation and
value);
        Read in colour filling style (fillstyle);
    Set up a workspace;
    Set up the connection to where the map, coverages and respective info files
        are located;
    Deal with ArcInfo coverages;
    Open the feature class "polygon";
    Pick up the layer's Attribute Table;
    Filter table's fields: FID, COLOUR, HUE, SATURATION, VALUE and FILLSTYLE only;
    Selection:
        the record where: FID = id#;
    Create a cursor on the previous selection;
    With the cursor, retrieve the selected record;
    Create auxiliary variables (C, H, S, V, and F) to store the values read in
        from the input file:
        C = colour;
        H = hue;
        S = saturation;
        V = value;
        F = fillstyle;
    Open an editing session;
```

```
    Update fields of the selected record:
        COLOUR = C;
        HUE = H;
        SATURATION = S;
        VALUE = V;
        FILLSTYLE = F;
    Store the updated record back in the layer's Attribute Table;
    Close the editing session;
End while
With each different value of the field COLOUR create a render of unique values:
    Create a cursor to loop through all records (field to be read: COLOUR only);
    Select all records of the Attribute Table;
    Loop through selected records:
        With the created cursor, retrieve COLOUR's value;
        If the value colour read is not the render yet Then
            Add colour to the list;
        Else
            Skip colour;
        End if
        If colour is a new value added to the render Then
            Create the associate symbol of the colour unique value:
                symbol's hue = H;
                symbol's saturation = S;
                symbol's value = V;
                symbol's filling style = F;
        End if
    End loop
With the created render of colour unique values & associate symbols:
    Redisplay the map features accordingly;
    Refresh map;
End
```

Attributes of arcs45copy polygon

Y-CENTER2	AREA2	PERIM2	SLOPECLASS	HUE	SATURATION	VALUE	FILLSTYLE	COLOUR
177314.484375	27.054199	131.770873	1	135	43	100	1	135043100
177156.015118	34305.215820	5177.044980	0	60	0	100	0	60000100
177311.045793	23.373535	41.120015	1	135	43	100	1	135043100
177271.988604	2111.659668	1126.497381	1	135	43	100	1	135043100
177312.095410	98.002197	102.985485	1	135	43	100	1	135043100
177314.28125	15.380615	65.273356	0	135	86	100	0	135086100
177313.598958	5.726074	28.886069	0	135	86	100	0	135086100
177312.746191	26.805664	26.073792	0	135	86	100	0	135086100
177312.974071	33.802002	35.615680	0	135	86	100	0	135086100
177313.507479	15.424072	53.733798	1	135	43	100	1	135043100
177304.437418	188.178711	134.651652	1	135	43	100	1	135043100
177308.339131	65.141113	56.240387	1	351	86	100	0	351086100
177309.123156	35.558350	48.255226	1	189	32	100	1	189032100
177191.695995	4150.387207	2026.755840	1	189	32	100	1	189032100
177309.973004	15.771729	16.042557	0	135	86	100	0	135086100
177312.053922	6.616943	13.147807	0	135	86	100	0	135086100
177312.453125	1.624512	7.411190	0	135	86	100	0	135086100
177310.854167	4.854980	10.185419	0	351	86	100	0	351086100
177311.644726	7.697510	13.691112	0	135	86	100	0	135086100
177304.941483	118.548828	70.476952	0	135	86	100	0	135086100

Figure 6.30 – Part of a PAT info file after the results of CFS and Rationalization are uploaded, and the respective fields populated. (The London case study)

As a final note, it should be pointed out also that while the interactive application is running, all the standard GIS tools are still active, and hence any current ArcMap functionality can always be combined, at any stage, with the application functionalities described above.

6.5 Summary

Starting from initially unstructured geospatial datasets of urban areas (*i.e.* no prior knowledge of the spatial entities is assumed), this chapter covered the steps undertaken in the design of a graph-theoretic approach, and showed how it could be applied in those circumstances towards the analysis of urban spatial topology.

The theoretical and practical methods designed to analyse entities within a LiDAR dataset of an urban environment were presented. In particular, the merits of depth-first and breadth-first search algorithms in analysing the structure of the urban spatial topology were discussed. Given the different ways both algorithms operate in traversing a graph, it was noted how BFS results are more meaningful in terms of the urban scene: the BFS tree branches are connected components of the original graph, and represent the shortest path between the root and their leaf; and it seems that they can be related to potential urban features.

Thus, the design of the graph analysis procedure was based upon BFS. It traverses the graph looking for sequential relationships of containment amongst the sequences of adjacency: the containment-first search (CFS). In fact, where containment occurs, there is a high likelihood of an urban feature being present.

However, a few preliminary experiments showed that developing CFS simply based on BFS is not sufficient for an effective interpretation of the urban scene topology. The CFS procedure had to be extended in order to be able to detect the spatial relation of containment in a broader sense. Indeed, there are some particular cases of containment which are not explicit in the original graph of adjacencies and have to be derived. In order to address this issue, the CFS algorithm was improved to be able to detect polygon-ring containments; this was implemented by taking into consideration the spatial relation of touching between steep polygons.

Details were given of how the graph-based method proposed was coded in ANSI C language. A series of flowcharts and algorithms were included as part of this chapter to illustrate how the constituent functions and procedures were developed and interconnected to build up the program.

The relevance of the visualisation aspects in the whole process of the urban scene analysis was also emphasised and discussed. To perform visual inspections, the incorporation of capabilities for the visual representation of both urban topology and its analysis, was proposed. In order to accomplish this, an interactive tool was implemented in ArcMap, using its embedded programming language, VBA, along with ESRI's ArcObjects. To illustrate which VB modules and user forms/objects were used, and how they were linked up, a diagram was included along with a series of screen shots to show how this application works. The two main procedures of the interactive tool were summarised in terms of pseudo code: uploading and drawing the graph of adjacencies spanning tree; updating the PAT info file with the results of CFS and Rationalization procedures, and the redisplay of the map of gradient regions.

It is pertinent to note once more that the interactive tool developed is a prototype of such a tool for the visualisation and inspection of urban topology. Given the GIS package used in this research, the interface implemented is undoubtedly characterised by the ArcGIS environment. Thus, it is acknowledged that, in a further step, the code should be modularised in order to be separated from the interface itself.

The algorithms implemented were also included in this chapter in the form of pseudo code. The associate code is printed out in the appendices of the thesis as indicated throughout this chapter. Likewise, examples of the output files of the different customised procedures are included in the respective appendices mentioned.

To summarise the analysis process of urban topology, the diagram depicted in Figure 6.31 takes an overview of the main steps described throughout Chapters 5 and 6, from initially unstructured data until higher level spatial structures are derived.

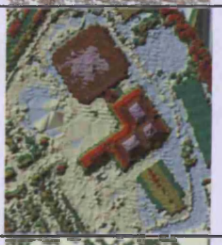
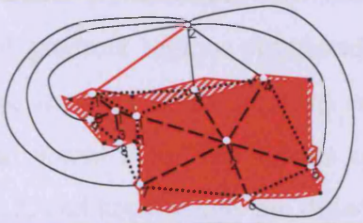
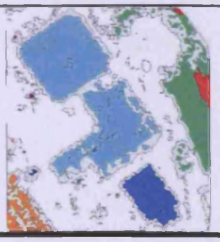
Start Unstructured data: a cloud of 3D points	
Data preparation (i) Topology is brought in: construction of a TIN for the internal description of the original point pattern	
Data preparation (ii) Binary classification of the TIN facets based on their gradient	
Establishment of a network of connectivity between polygonal regions Construction of polygon adjacencies graph & steep-polygon touchings graph	
Containment-first search Analysis of the adjacencies graph structure & Detection of the polygon-ring containments by simultaneously analysing the touchings graph	
Finish Derivation of higher level spatial structures: containment units & Dynamic display of the containment units identified in the original map of polygonal regions	

Figure 6.31 - Summary of the analysis process of urban topology, from start (unstructured data) to finish (containment units).

Before experiments with real world data are presented and discussed, for the sake of proof of concept the next chapter covers the application of the graph-theoretic approach proposed to simulated data.

7 Proof of concept

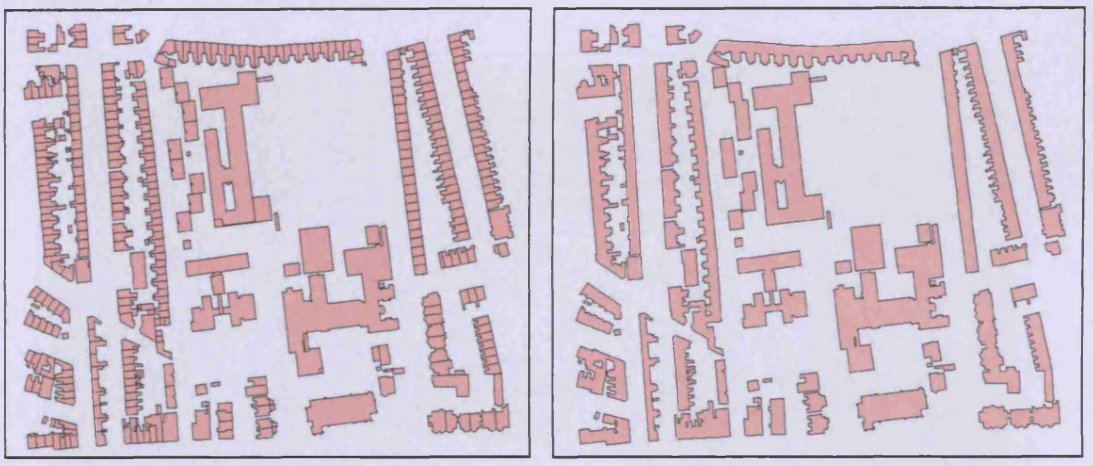
Before tests with real initial unstructured data of urban systems are undertaken, this chapter describes an experiment carried out with synthetic idealised spatial data relating to urban objects. The main aim is to explicitly show, in the absence of noise and error, the utility of the functionality of the algorithm presented in the previous chapter; *i.e.* the algorithm developed do indeed show that spatial topology and in particular containment relationships relate to the so called higher-level urban scene objects (typically, sets of different buildings).

The layout of this chapter is as follows. Section 7.1 describes how idealised noise-free data were simulated. Given the map of gradient regions simulated, section 7.2 presents the associate graph of adjacencies, its main characteristics and how they differ from those of graphs relating to real world data. The results of the spatial topology analysis are presented in section 7.3, and conclusions are drawn in terms of the assertions made when designing the algorithms. Finally, a summary of this chapter is given in section 7.4.

7.1 Generation of synthetic data

For the purpose of this experiment, idealised data, free of noise and error, had to be generated. A map of binary classified gradient regions was created simulating a map of higher-level urban scene objects. This was done by initially taking some building

polygons from OS Master Map data[10] (*vd.* Figure 7.1a). These include several kinds of buildings such as terraced, detached and semidetached houses. Building polygons sharing arcs were then dissolved in order to simulate higher-level structures (*vd.* Figure 7.1b). These polygons were taken to play the role of the flat polygons, and were classified accordingly.



a) b)

Figure 7.1 – Building footprints taken from OS Master Map data[10]: a) single building polygons; b) higher-level structure polygons.

In order to simulate the steep polygons, the flat polygons above were then buffered: $1m$-buffer polygons were generated around flat polygons in a second level of information within the ArcMap project.

For topological reasons, a third entity needed to be considered. An outer polygon, distinct from the Universe Polygon, was created in a third level of information. This polygon (the external rectangle in Figure 7.2) was necessary to delimitate the whole area and to simulate the ground polygon.

Then, the GIS spatial operation of overlay (union) was carried out in ArcMap between the three levels of information described above. The result of this operation was a simulated map of 150 gradient regions (steeps – "Slopeclass"= 1; and flats – "Slopeclass" = 0) pictured in Figure 7.2. Recall that the ground polygon is assumed

[10] Made available by the Department of Geomatic Engineering of the University College London for academic purposes. Ordnance Survey ©Crown Copyright, all rights reserved.

to be a flat region. As Figure 7.2 shows, steep polygons shape both buildings standing on their own and higher-level structures; the enclosed flat polygons simulate building roofs; in some instances, some complexity was brought in simulating structures on top of buildings – this was done by adding a few more rings of steep polygons enclosing flat polygons (*vd.* example highlighted by yellow ellipse in Figure 7.8).

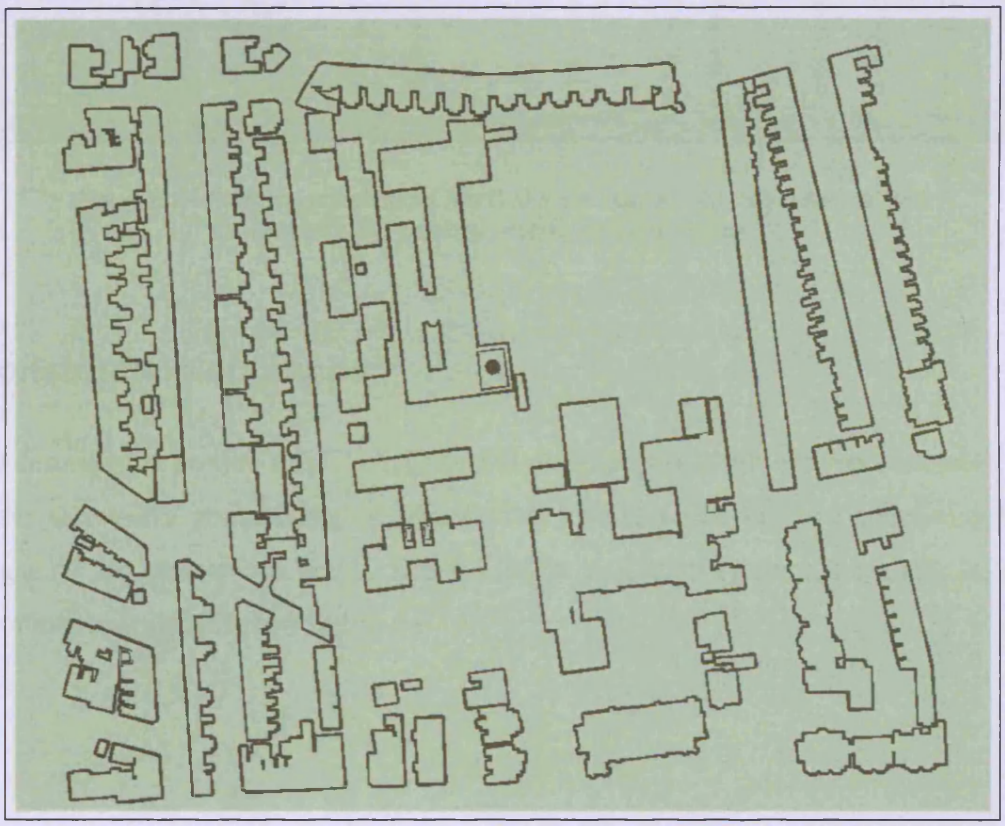


Figure 7.2 – Simulated map of gradient regions: binary classification of the polygons generated into steep polygons (in dark green) and flat polygons (in light green).

In turn, Figure 7.3 depicts an extract of the associate polygon attribute table (PAT) before spatial topology was analysed. As shown, the binary classification of gradient regions can be seen in column "Slopeclass"; "Hue", "Saturation", "Value", "Colour" and "Fillstyle" fields are not populated until spatial topology analysis is undertaken.

Attributes of urbanfeatures polygon

FID	Shape	AREA	PERIMETER	URBANFEATURES	URBANFEATURES-I	SLOPECLASS	HUE	SATURATION	VALUE	COLOUR	FILLSTYLE
2	Polygon	70834,908845	9356,246896	2	0	0	0	0	0	0	0
3	Polygon	64,935917	129,880173	3	0	1	0	0	0	0	0
4	Polygon	122,493821	241,916612	4	0	1	0	0	0	0	0
5	Polygon	150,127299	63,186314	5	0	0	0	0	0	0	0
6	Polygon	61,364073	122,733908	6	0	1	0	0	0	0	0
7	Polygon	55,252188	35,763519	7	0	0	0	0	0	0	0
8	Polygon	121,46816	58,879522	8	0	0	0	0	0	0	0
9	Polygon	31,722502	63,446256	9	0	1	0	0	0	0	0
10	Polygon	479,360344	949,778515	10	0	1	0	0	0	0	0
11	Polygon	141,906543	61,623635	11	0	0	0	0	0	0	0
12	Polygon	45,808288	28,794393	12	0	0	0	0	0	0	0
13	Polygon	1144,999075	409,367081	13	0	0	0	0	0	0	0
14	Polygon	644,010792	1279,130219	14	0	1	0	0	0	0	0
15	Polygon	1586,758884	513,792094	15	0	0	0	0	0	0	0
16	Polygon	821,009071	1634,73704	16	0	1	0	0	0	0	0
17	Polygon	1385,698496	486,51905	17	0	0	0	0	0	0	0
18	Polygon	24,55598	22,651218	18	0	0	0	0	0	0	0
19	Polygon	10,869064	13,605703	19	0	0	0	0	0	0	0
20	Polygon	1305,102492	337,288977	20	0	0	0	0	0	0	0
21	Polygon	12,292974	19,253779	21	0	0	0	0	0	0	0
22	Polygon	0,194633	2,01571	22	0	0	0	0	0	0	0
23	Polygon	2,342642	7,18254	23	0	0	0	0	0	0	0
24	Polygon	54,803781	109,615116	24	0	1	0	0	0	0	0
25	Polygon	118,828203	52,75528	25	0	0	0	0	0	0	0

Record: 1 Show: All Selected Records (0 out of 150 Selected) Options

Figure 7.3 – Polygon attribute table (PAT) of the simulated map of gradient regions before spatial topology analysis was undertaken.

7.2 Construction of graphs

As described in section 6.2.1, polygons and associate arc attributes were accessed in order to retrieve gradient-region adjacencies. The text file obtained containing the edges of the adjacencies graph, was loaded in to Ucinet system; the graph layout obtained is depicted below in Figure 7.4.

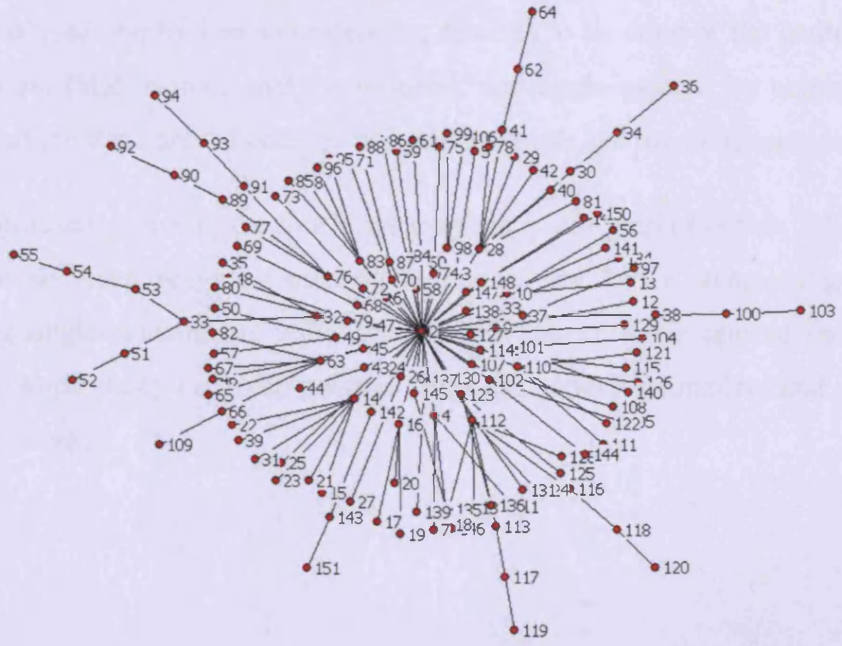


Figure 7.4 – Graph of adjacencies of the simulated map of gradient regions. (Generated using Ucinet 6 for Windows; Borgatti et al.*, 2002)*

Vertex *2*, relating to the simulated ground polygon, is the most connected polygon (*i.e.* the UEB) and can be seen right in the centre of the graph layout. The "cloud" of vertices surrounding vertex *2* represents the first level of adjacency. Note that, in this case, this also represents the first level of containment; indeed, those vertices surrounding vertex *2* refer to the steep polygons adjacent to the ground polygon which are also contained within it. The outer ring of vertices in the graph layout represents the second level of adjacency; each vertex relates to a flat polygon contained within a single steep polygon at the first level of adjacency/containment mentioned earlier. Thus, as before, the second level of adjacency is also a level of containment. And so on so forth, *i. e.* the nine longer paths that can be counted around the outer cloud of vertices basically represent particular situations of flat polygons enclosing other rings of steeps and their enclosed flats. These could be referring, for instance, to structures on top of buildings.

The main characteristic of the adjacencies graph obtained when dealing with clean data is the fact that all relationships of adjacency are also relationships of containment. Hence, the containment search process and detection of containment units are more straightforward tasks. This is the most complex aspect when dealing with real world data: as seen in Chapter 6, it is not guaranteed before hand that a spatial relation of adjacency is also of containment. In fact, spatial relationships can not be analysed one by one separately but this has to be done in the context of the whole scene (higher-order analysis process), and made explicit by combining the analyses of the structures of both the adjacencies graph and the touchings graph.

In this idealised situation, the touchings graph is a null graph (section 3.2). Indeed, there are no steep polygons meeting at nodes, and hence steep-polygon rings, enclosing single containment units, are not split into different entities (as happens with real world data); *i.e.* steep-polygon rings are perfectly complete and consist of only one polygon.

7.3 Spatial topology analysis

Figure 7.5 depicts the results of the urban topology analysis for the simulated map of gradient regions. Polygon *2* (mapped in white), with 45 adjacent regions, and corresponding to the ground polygon, was chosen as the UEB.

By visually inspecting the map of containment units obtained, one can see that the algorithm performed according to what it was designed for. Indeed, the algorithm individually detected all the urban features simulated as separate containment units. Moreover, individual simulated spatial features closely standing next to one another, but not actually juxtaposed, were detected separately. This confirms that in theory the algorithm is capable of detecting single buildings standing on their own.

Because this is noise-free data, chains of steep polygons enclosing containment units are not open and consist of only one polygon; these also outline the associate spatial feature. In addition to this, there are no map edge effects.

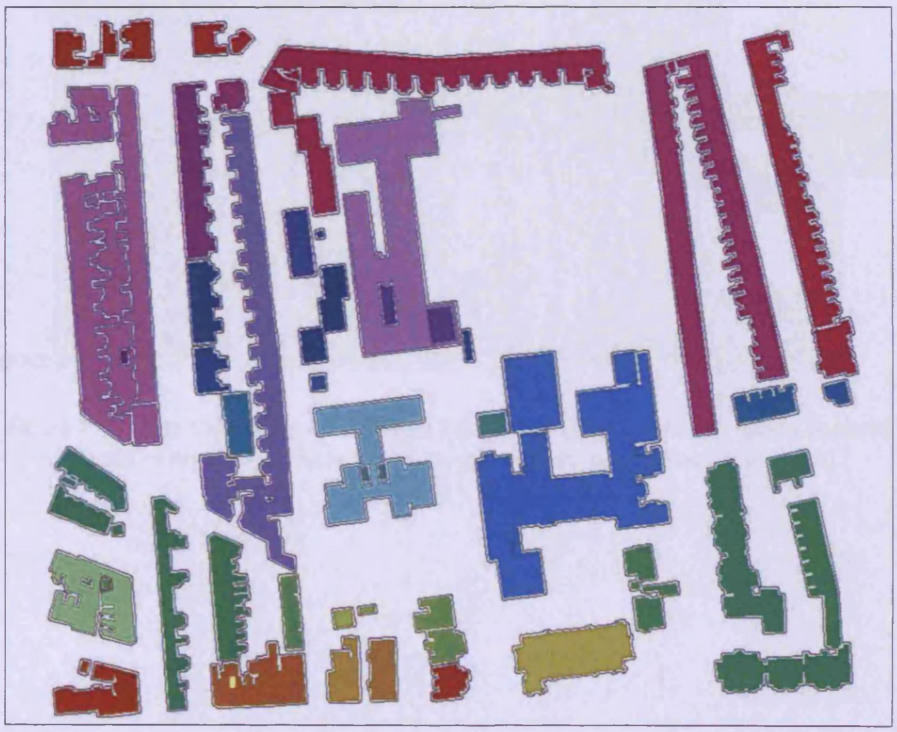


Figure 7.5 – Spatial topology analysis and the different containment units identified.

The results of the experiment with synthetic data confirmed the assumption that each branch of the breadth-first search tree directly relates to a spatial feature (*vd.* example

in Figure 7.6). In some more complex cases, like the one highlighted by the yellow and red ellipses in Figure 7.6, possible sub-branches refer to sub-containment units within the main one; but the whole branch still relates to a single containment unit.

Furthermore, the results obtained also demonstrated that the algorithms developed do show the spatial topology. In fact, sequences of adjacencies/containments were correctly detected as so by the algorithm, and also correctly represented by the interactive tool (*vd.* Figure 7.7 – as described earlier, solid colours correspond to flat polygons, and coloured hashed patterns correspond to steep polygons; the higher the level of adjacency/containment of polygons within a given unit, the darker the tone of the colour assigned to it). The results also confirmed that containment relationships do relate to higher-level urban scene features.

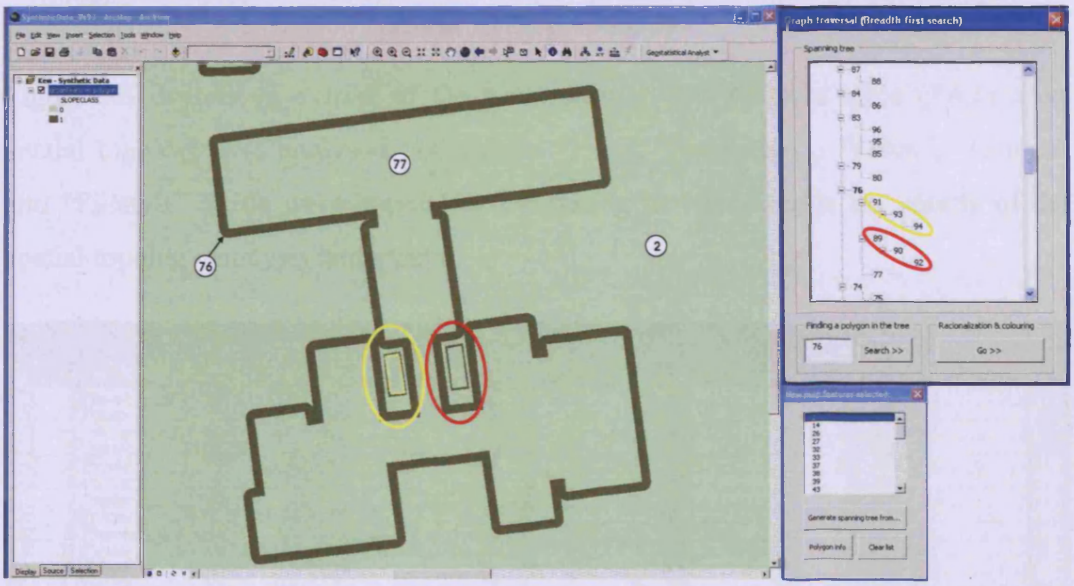


Figure 7.6 – Branches of the breadth-first search tree directly relate to spatial features. (Detail of Figure 7.2; the numbers on the map are polygon labels in insert)

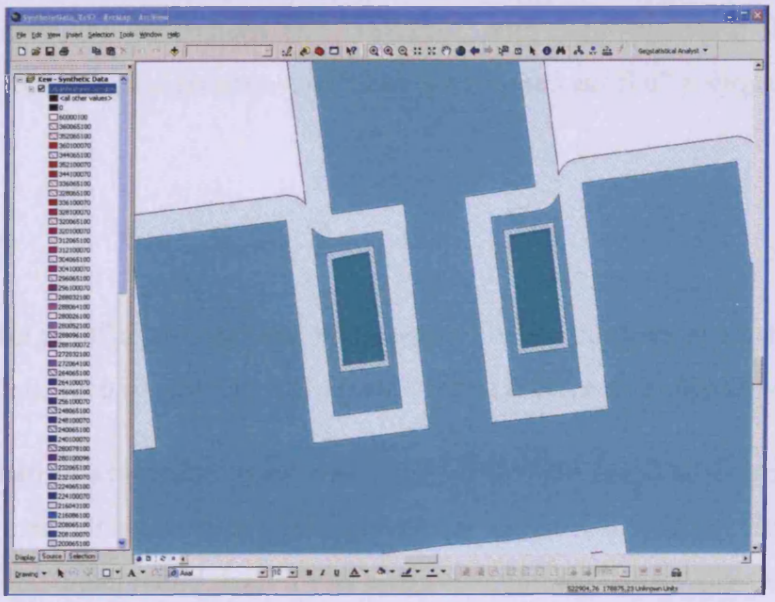


Figure 7.7 – Sequences of containments correctly detected within a given containment unit. (Detail of Figure 7.6 above)

Figure 7.8 depicts an extract of the associate polygon attribute table (PAT) after spatial topology was analysed. As shown, "Hue", "Saturation", "Value", "Colour" and "Fillstyle" fields were populated, translating in visual terms the results of the spatial topology analysis undertaken.

Attributes of urbanfeatures polygon

FID	Shape *	AREA	PERIMETER	URBANFEATURES	UR	SLOPECLASS	HUE	SATURATION	VALUE	COLOUR	FILLSTYLE
2	Polygon	70834.908845	9356.246896	2	0	0	60	0	100	60000100	0
3	Polygon	64.935917	129.880173	3	0	1	360	65	100	36006510	1
4	Polygon	122.493821	241.916612	4	0	1	352	65	100	35206510	1
5	Polygon	150.127299	63.186314	5	0	0	360	100	70	36010007	0
6	Polygon	61.364073	122.733908	6	0	1	344	65	100	34406510	1
7	Polygon	55.252188	35.763519	7	0	0	352	100	70	35210007	0
8	Polygon	121.46816	58.879522	8	0	0	344	100	70	34410007	0
9	Polygon	31.722502	63.445256	9	0	1	336	65	100	33606510	1
10	Polygon	479.360344	949.778515	10	0	1	328	65	100	32806510	1
11	Polygon	141.906543	61.623635	11	0	0	352	100	70	35210007	0
12	Polygon	45.808288	28.794393	12	0	0	336	100	70	33610007	0
13	Polygon	1144.999075	409.387091	13	0	0	328	100	70	32810007	0
14	Polygon	644.010792	1279.130219	14	0	1	320	65	100	32006510	1
15	Polygon	1586.758884	512.792094	15	0	0	320	100	70	32010007	0
16	Polygon	821.009071	1634.73794	16	0	1	312	65	100	31206510	1
17	Polygon	1385.698496	486.51905	17	0	0	312	100	70	31210007	0
18	Polygon	24.55598	22.651219	18	0	0	352	100	70	35210007	0
19	Polygon	10.869064	13.605703	19	0	0	312	100	70	31210007	0
20	Polygon	1305.102492	337.288977	20	0	0	312	100	70	31210007	0
21	Polygon	12.292974	19.253779	21	0	0	320	100	70	32010007	0
22	Polygon	0.194633	2.01571	22	0	0	320	100	70	32010007	0
23	Polygon	2.342642	7.18254	23	0	0	320	100	70	32010007	0
24	Polygon	54.803781	109.615116	24	0	1	304	65	100	30406510	1
25	Polygon	118.828203	52.75528	25	0	0	304	100	70	30410007	0

Record: 1 Show: All Selected Records (1 out of 150 Selected) Options

Figure 7.8 – Polygon attribute table (PAT) of the simulated map of gradient regions after spatial topology analysis was undertaken.

Because there are no steep-polygon islands within flat polygons in this noise-free environment, "Fillstyle" values equal "Slopeclass" values; in other words, all the steep polygons were mapped as so. Recall that, when dealing with real data, steep-

polygon islands are generalised and "merged" with their enclosing flat polygon during the rationalization process, and hence are mapped as "flat" polygons.

7.4 Summary

In this chapter proof of concept was undertaken. For the purpose, synthetic data was generated, and a map of gradient regions simulating urban scene objects was created.

Given the map of gradient regions created, the associate graph of adjacencies was derived. Its main characteristics were pointed out and how they differ from those of graphs relating to real world data. It was also noted and explained why that in such a simulated environment, free of noise and error, the steep-polygon graph of touchings may be a null graph.

The analysis of the spatial topology was undertaken, and conclusions were drawn in terms of the assertions made when designing the algorithms. The results obtained demonstrated that in the absence of noise and error the algorithms do indeed show the spatial topology. In particular, the results support the assumption that each breadth-first search tree's branch does relate to a single containment unit within the initial map of gradient regions. Moreover, sequences of containment relationships do relate to higher-level urban scene objects.

The next chapter will present and discuss the results obtained by applying the methodology implemented to real world data. Since the London (Kew) dataset was primarily used for development purposes of the methodology, the emphasis will be given in particular to the results obtained with the Stuttgart LiDAR datasets. In addition, the implications of working with real initial unstructured datasets of urban systems will be commented and discussed.

8 Validation and discussion

The results obtained by applying the algorithm developed to real world data are presented and discussed in this chapter. Since the London dataset was primarily used for development purposes of the methodology proposed, the emphasis is given in particular to the results obtained for the Stuttgart LiDAR datasets (including two different sites – a city centre area, and a predominantly residential area). This chapter is structured as follows.

The experiments carried out with the different sources of data are separated into two sections: the experiments with LiDAR data are presented and discussed in section 8.1; section 8.2 presents and discusses the experiments accomplished with other sources of data, namely photogrammetric data.

The first experiments with LiDAR data used raw data acquired in the surveyed areas of London (Kew) and Stuttgart; these are included in section 8.1.1. Morphologically filtered data, obtained from the original raw data, were also available in the Department of Geomatic Engineering of the UCL for the same surveyed areas. Thus, further experimentation was carried out using morphologically filtered data whose results are presented in section 8.1.2.

The experiments with photogrammetric data are described in section 8.2. Three datasets of three different areas surveyed were mainly used: Barton-Bendish (in the UK), Heerbrugg (Switzerland), and Santa Lucija (Malta).

A statistical evaluation of key results is carried out in section 8.3. Reference data are presented, and the process to generate them is described in this section. Also defined

are the quality measures used in this thesis for a quality assessment based upon a comparison of coincidences in areas.

A summary and a general discussion of this chapter is given in section 8.4.

8.1 Experiments with LiDAR data

8.1.1 Raw data

8.1.1.1 The London dataset

After the experiments performed for the case study area of Kew (London), the first experiment carried out entailed the application of the algorithm to the whole dataset of Kew. The same 45° gradient threshold was used, and the results obtained are shown below in Figure 8.1. The map of steep/flat polygons comprises 2933 polygons (*vd.* Figure 5.12b).

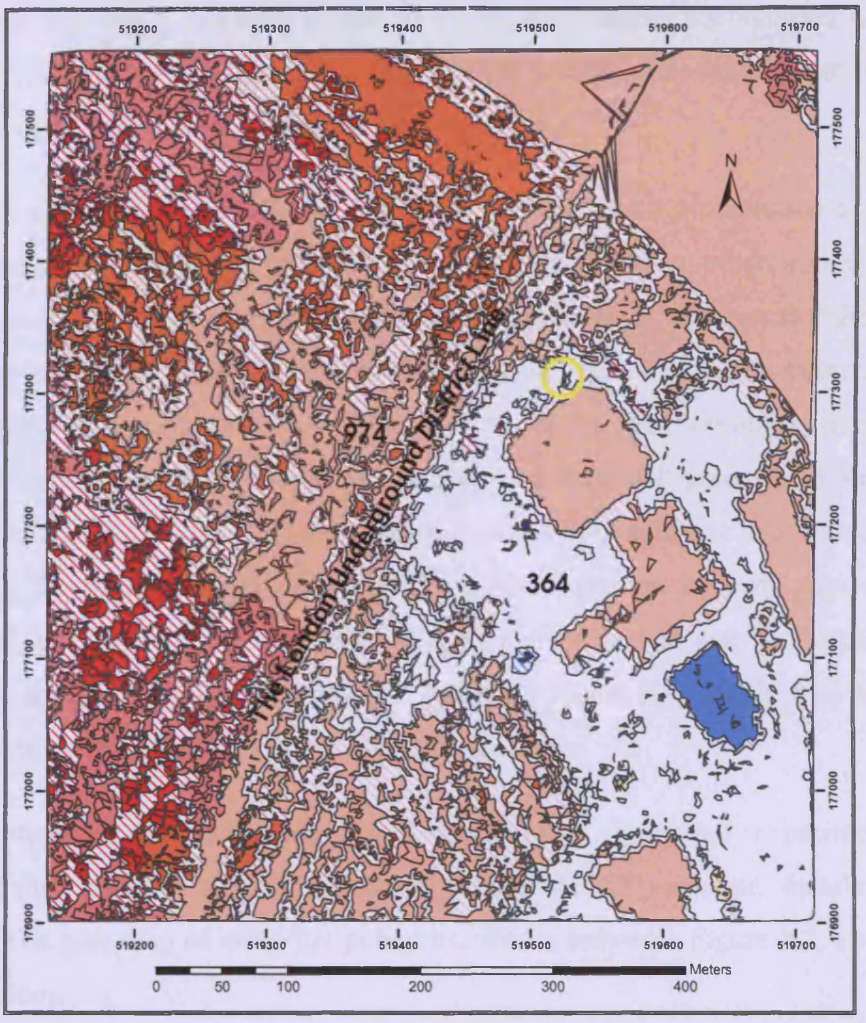


Figure 8.1 –The urban topology analysis and the different containment units identified – using the 45° gradient threshold.
(The Kew, southwest London, dataset)

The first aspect to point out is that the railway, crossing the river Thames and running south-westward across the dataset, split the potential UEB into different polygons. Amongst them, the largest two are located on both sides of the railway: polygon *364* on the east side (with 146 adjacencies), and polygon *974* on the west side (with 324 adjacencies) (*vd.* Figure 8.1). The ground polygon surrounding the National Archives building (polygon *364*, mapped in white) was chosen as the graph search root.

The whole area on the west side of the railway was identified as part of the same containment unit. This unit was linked practically to the whole area on the east side too along the map edges: through the river Thames in the north part, and through the residential area south of the National Archives. In fact, apart from the building

mapped in blue and a couple of bushes and trees surrounding the National Archives, the surveyed area was practically identified as part of the same containment unit, and hence mapped in the same orange colour (Figure 8.1).

It is believed that the main reasons for this fact are both the low resolution of the data (3m point spacing), and the complex pattern of this particular urban scene. Indeed, the west side of the surveyed area covers a residential zone with small buildings (if compared with the National Archives building), and with vegetation taller than the actual buildings in most of the cases. Because of the low data resolution, most of the urban features are not well shaped by the chains of steep polygons; these turned out to be linked up, assembling different urban features into only one containment unit. Although the National Archives building clearly stands on its own, this was not identified as a separate containment unit either. This fact is due to a small steep polygon (north of the building, circled in yellow in Figure 8.1) which links the steep polygon ring enclosing the building to rest of the data.

As an attempt to obtain more meaningful results, a second experiment was accomplished using a slightly higher threshold; the 50° gradient threshold was chosen. The new map of steep/flat polygons, shown below in Figure 8.2, comprises 2914 regions.

Figure 8.2 – The binary classification of the TIN facets and the generation of steep/flat polygons – using the 50° gradient threshold.
(The Kew, southwest London, dataset)

The results of the urban topology analysis can be seen in Figure 8.3. Polygon *5*, with 1023 adjacencies, was chosen as the UEB (mapped in white).

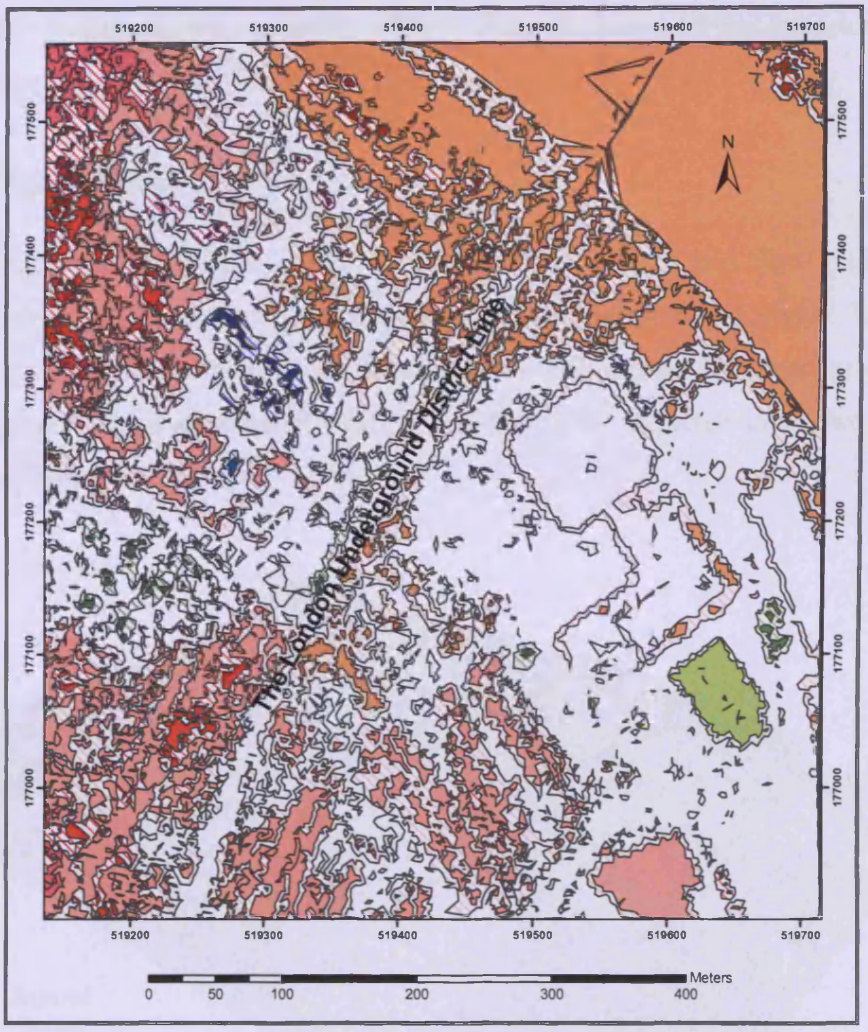


Figure 8.3 - The urban topology analysis and the different containment units identified – using the 50° gradient threshold.
(The Kew, southwest London, dataset)

By raising the gradient threshold some of the noise effect was eliminated. It can be seen that the UEB considerably extended across the railway to include most of the western area.

More containment units were identified separately now; however, the results are still meaningless, especially on the west side of the surveyed area. The most important fact to be noted is that the 50° gradient value is still a very high threshold for this dataset. In fact, when dealing with the case study area, it had been concluded already that a 60° gradient thresholding was also very high (*vd.* Chapter 5).

The idea of applying the algorithm to a different dataset with a higher resolution, covering the same area, was thought of. However, due to various reasons, it was not

possible to arrange higher resolution LiDAR data, and hence further experimentation was not accomplished for this area.

8.1.1.2 The Stuttgart datasets

Both datasets covering Site 1 (an area in the city centre) and Site 2 (a mostly residential area) in Stuttgart were pre-processed as described in Chapter 5, section 5.4.2. Topological information was brought in to each dataset by generating a TIN. The associate DSMs obtained for Site 1 and Site 2 are depicted in Figure 8.4 and Figure 8.5 respectively.

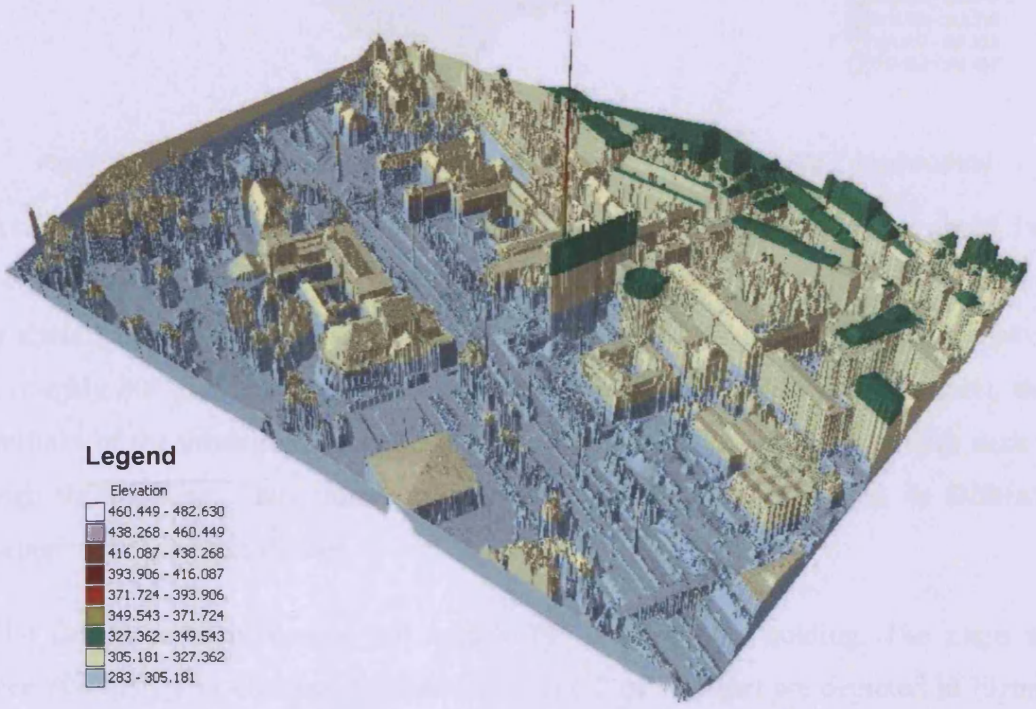


Figure 8.4 – The DSM generated from the LiDAR point set of Stuttgart – Site 1 (original data).

An artefact can be seen in the middle of the DSM above. This fact could have been caused by a bird which was hit by a LiDAR beam. As a consequence, the point elevation range was extended up to 482m; however, the actual maximum height of the urban features does not go beyond 340m approximately.

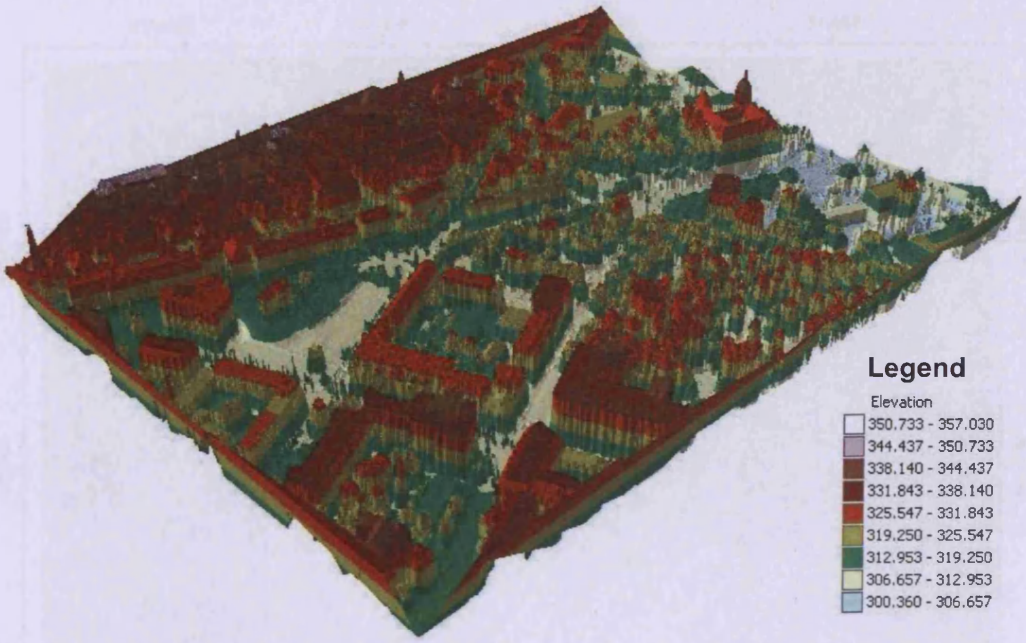


Figure 8.5 – The DSM generated from the LiDAR point set of Stuttgart – Site 2 (original data).

As explained in Chapter 5, given the fact that the Stuttgart datasets are about 1m point spacing on the plane, and supposing that the average height of an urban feature is about 5m, a TIN facet straddling an urban feature and the local terrain should have a roughly 80° gradient. However, as happened with the London (Kew) dataset, the outlines of the urban features are not well defined by the steep regions with such a high thresholding. Thus, lower gradient thresholds were considered in different experiments with this dataset.

The first experiment carried out used a 70° gradient thresholding. The maps of steep/flat polygons obtained for Site 1 and Site 2 of Stuttgart are depicted in Figure 8.6 and Figure 8.7 respectively. The Site 1 map comprises 9415 steep/flat polygons, and the Site 2 map comprises 8092 regions.

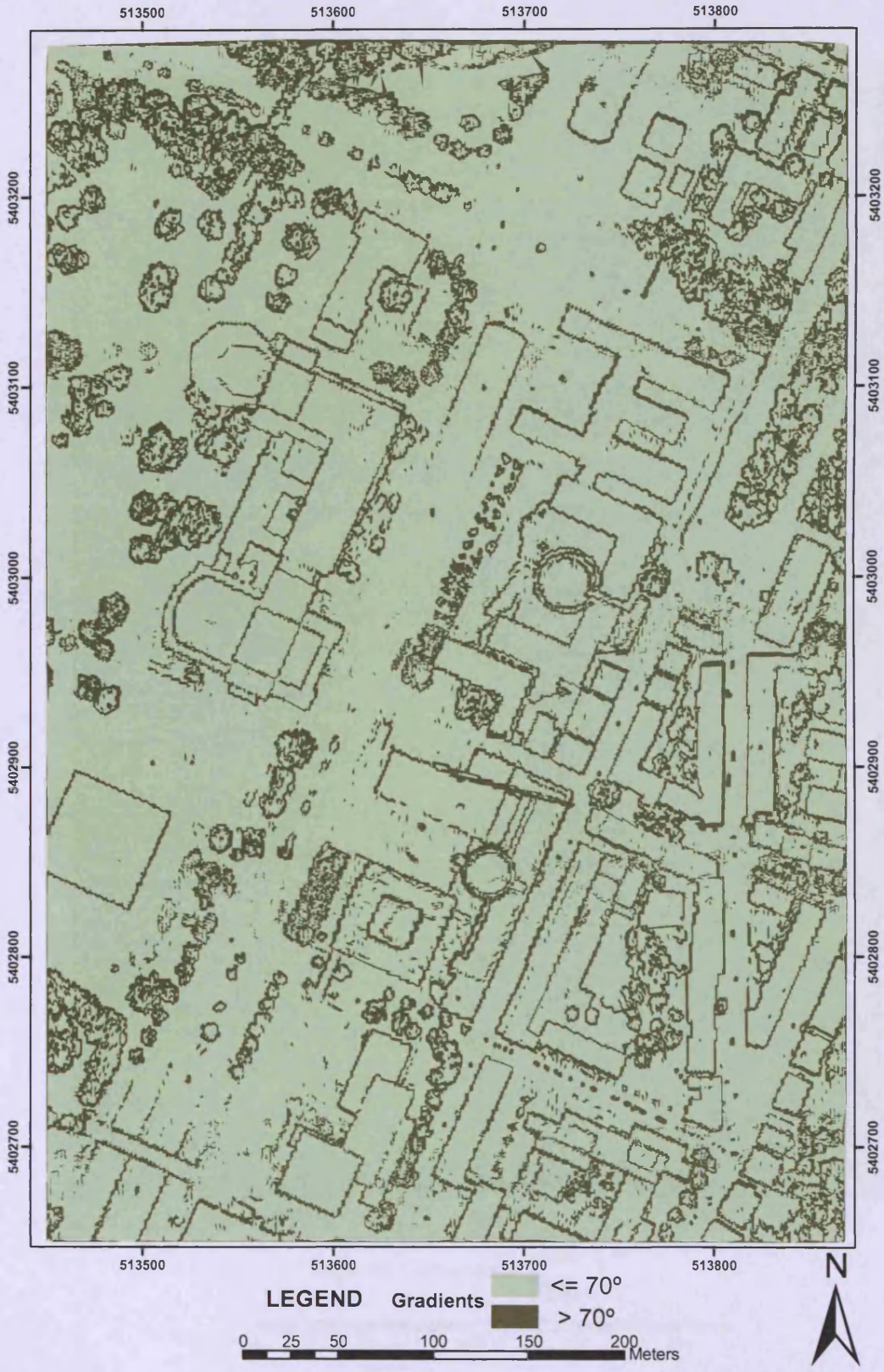


Figure 8.6 – The binary classification of the TIN facets and the generation of steep/flat polygons – using the 70° gradient threshold.
(The Stuttgart dataset, Site 1)

Figure 8.7 – The binary classification of the TIN facets and the generation of steep/flat polygons – using the 70° gradient threshold.
(The Stuttgart dataset, Site 2)

Figure 8.8 depicts the results of the urban topology analysis for Site 1. The chosen UEB was polygon *35* (mapped in white) with 3056 adjacent regions.

It can be seen that the main blocks of buildings and sets of vegetation in Site 1 were identified by the algorithm as separate containment units. There are two main exceptions that need to be pointed out. The buildings and vegetation lying along the map edge were mapped with the same colour because the algorithm identified them as part of the same containment unit. Edge effects are the reason for this fact; indeed, the sliver steep polygons around the map border ended up gathering all those different urban features into one unit.

In addition, the three buildings circled in yellow, in the south, were wrongly considered as part of the same containment unit. This was caused by the vegetation meandering around them, in particular the set of trees next to the building in the very south. In fact, this set of trees established a connection between the three buildings, and all these spatial features turned out to be one.

A few stripes of tiny steep polygons can also be seen horizontally across the map depicted in Figure 8.8. This is a consequence of LiDAR swath overlap. If these steep polygons happen to lie within flat regions, they become one with the respective enclosing flat region during the rationalization process. As explained in Chapter 6, the borders of the steep-polygon islands are still mapped for illustration purposes in order to show the extent of the generalisation task.

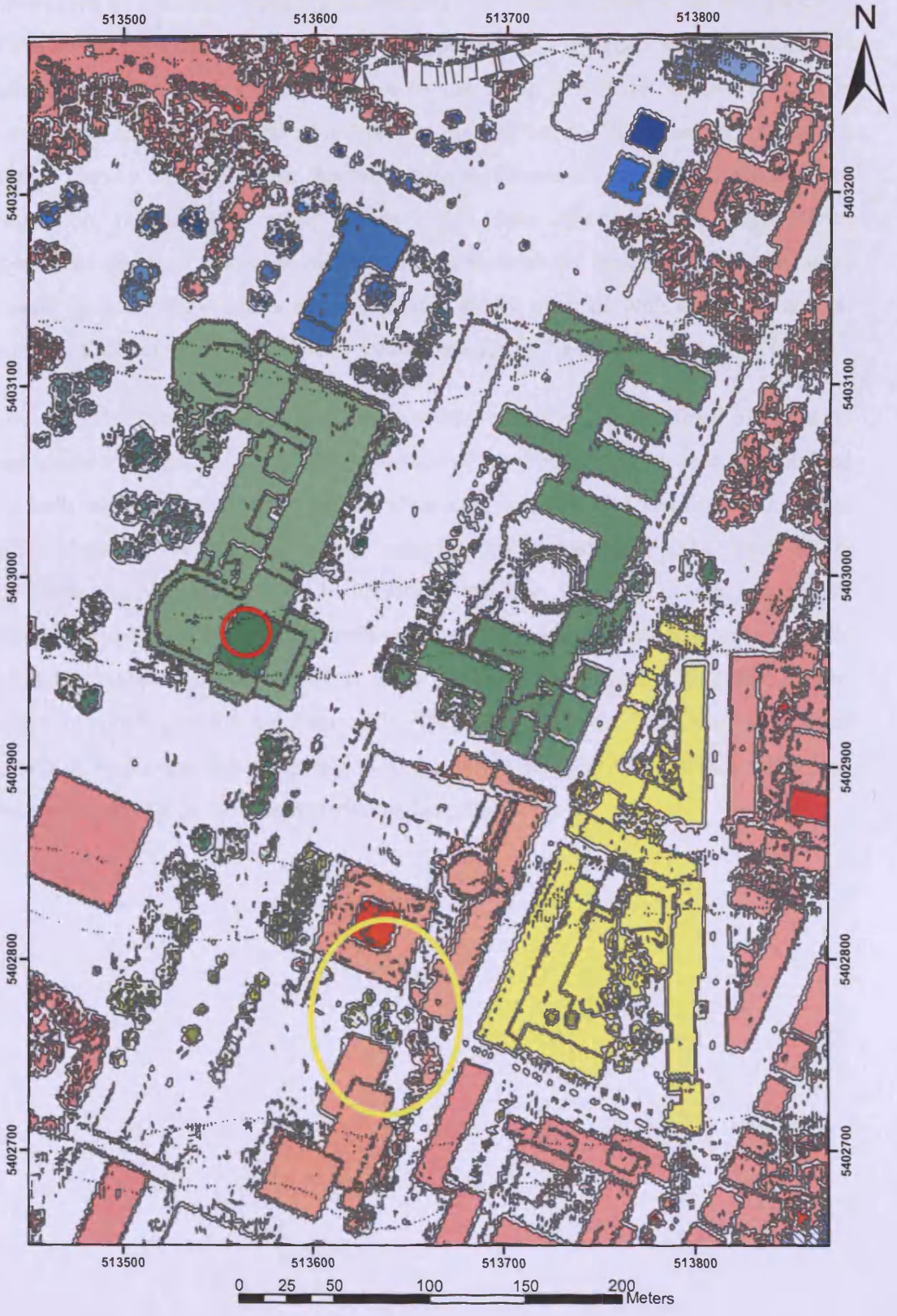


Figure 8.8 - The urban topology analysis and the different containment units identified – using the 70° gradient threshold.
(The Stuttgart dataset, Site 1)

The results of the urban topology analysis for Site 2 are depicted below in Figure 8.9. The root of the graph search process was polygon *23* (mapped in white) with 934 adjacencies. Site 2 represents a more complex urban pattern for it covers a mostly residential area. If compared with Site 1 in the city centre, Site 2 has in particular a higher density of small urban features, such as houses surrounded by all kinds of vegetation, ranging from small bushes to tall trees. Because of the edge effects mentioned above, a couple of different urban features on the edge of the map were linked up in a single containment unit, and hence mapped with the same colour. Some of these links are indicated by yellow crosses in Figure 8.9.

These two Stuttgart datasets constitute raw data which were not filtered in any way, and hence are considerably noisy. The results of the urban topology analysis obtained for both sites show that, even so, the algorithm is capable of identifying the main urban blocks in the surveyed area as separate containment units. The depth of the associate spanning tree is 6 both in Site 1 and Site 2, which means 6 levels of adjacency/containment in both cases. In most of the instances, the last level of adjacency/containment is related to noise (*vd.* some examples highlighted in red circles in both Figure 8.8 and Figure 8.9). They represent most likely small structures on top of buildings, like chimneys, antennas or air conditioning systems, which are not really relevant in the interpretation of the urban scene.

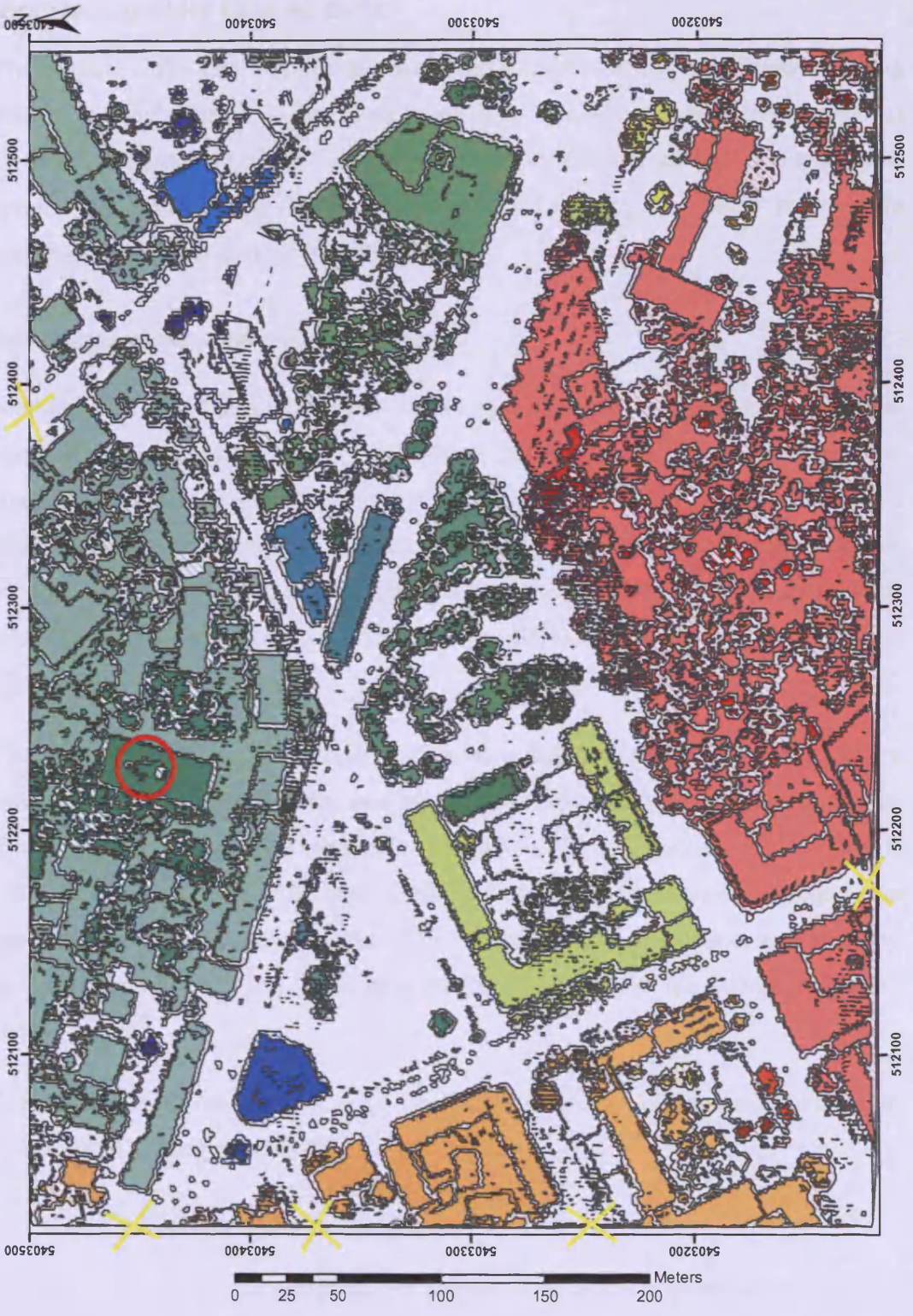


Figure 8.9 - The urban topology analysis and the different containment units identified – using the 70° gradient threshold.
(The Stuttgart dataset, Site 2)

8.1.2 Morphologically filtered data

The original LiDAR data of Stuttgart had been subjected to a morphological filtering process (Ayugi, 2006). These data were also available at the Department of Geomatic Engineering of the University College London. Thus, further experiments were accomplished using filtered data in order to investigate whether it would be possible to obtain more meaningful results.

8.1.2.1 Generalities about morphological filtering

Mathematical morphology has been widely used in image processing. It composes operations based on set theory to extract features from an image, and two fundamental operations can be identified: *dilation* and *erosion* (Zhang *et al.*, 2003). These operations are commonly used to enlarge (dilate) and reduce (erode) the size of features in binary images. Dilation and erosion are often combined to produce *opening* and *closing* operations (Zhang *et al.*, 2003), which are here employed to filter LiDAR data.

The concept of dilation and erosion can be extended from the binary case to find a minimum and a maximum value in a specified window kernel in grey scale images. These concepts can also be extended to the analysis of a continuous surface such as a DSM as measured by LiDAR data (Zhang *et al.*, 2003). In image processing, the grey scale values are used as input for the morphological filtering; when applying morphological filtering to LiDAR data, the 3D coordinates of the LiDAR points are used as input.

For a LiDAR measurement $p(x, y, z)$, the dilation of the height z at the (x, y) location is defined as (Zhang *et al.*, 2003):

$$d_p = \max_{(x_p, y_p) \in w} (z_p), \qquad \textit{(Equation 8.1)}$$

where points (x_p, y_p, z_p) represent p's neighbours (by their coordinates) within a window, w. The window can be a one dimensional line, or a two-dimensional rectangle, square, circle, ellipse, etc. The dilation output is the maximum height value in the neighbourhood of p.

The erosion operation is the counterpart of dilation and is defined as (Zhang *et al.*, 2003):

$$e_p = \min_{(x_p, y_p) \in w} (z_p), \qquad \textit{(Equation 8.2)}$$

The combination of dilation and erosion generates the *opening* and *closing* operations that are employed to filter LiDAR data (Zhang *et al.*, 2003). The opening operation is achieved by performing an erosion of the dataset followed by a dilation, whilst the closing operation is accomplished by carrying out a dilation first and then an erosion. In the opening operation, typically the erosion eliminates all local maxima in height, whilst the dilation restores the shape of the terrain surface (Zhang *et al.*, 2003).

8.1.2.2 The Stuttgart datasets

The morphologically filtered datasets relating to Site 1 and Site 2 were also pre-processed as described in Chapter 5, section 5.4.2. Topological information was brought in to each dataset by generating a new TIN. The new DSMs obtained for Site 1 and Site 2 with morphologically filtered LiDAR data are depicted in Figure 8.10 and Figure 8.11 respectively.

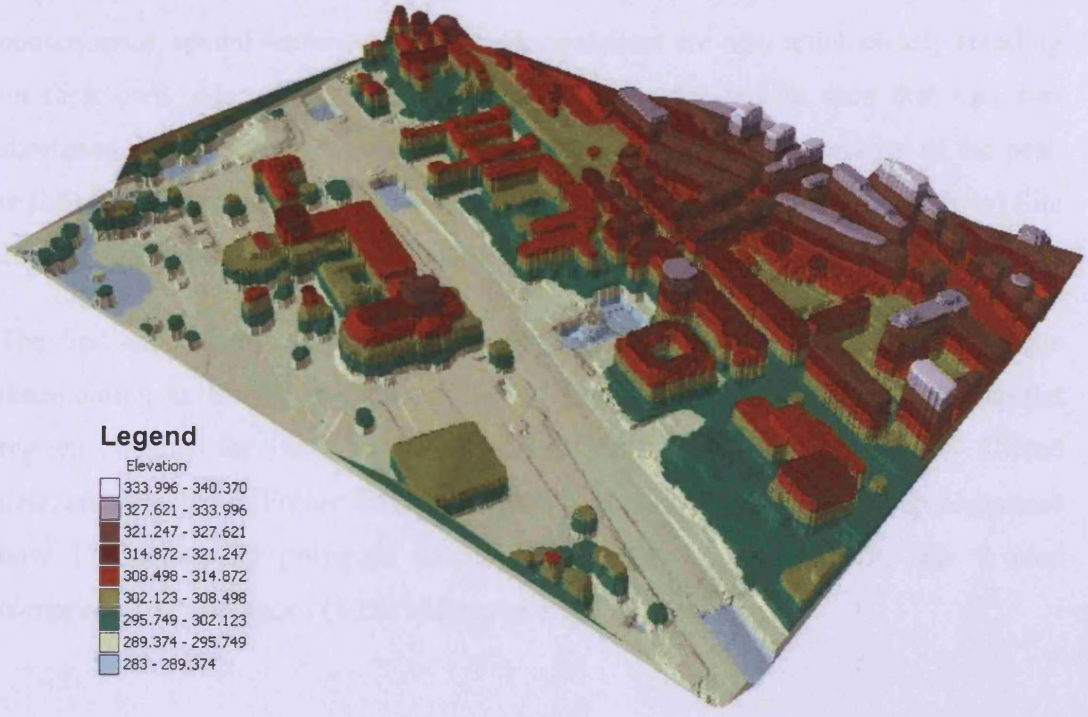


Figure 8.10 – The DSM generated from the morphologically filtered LiDAR data - Stuttgart, Site 1.

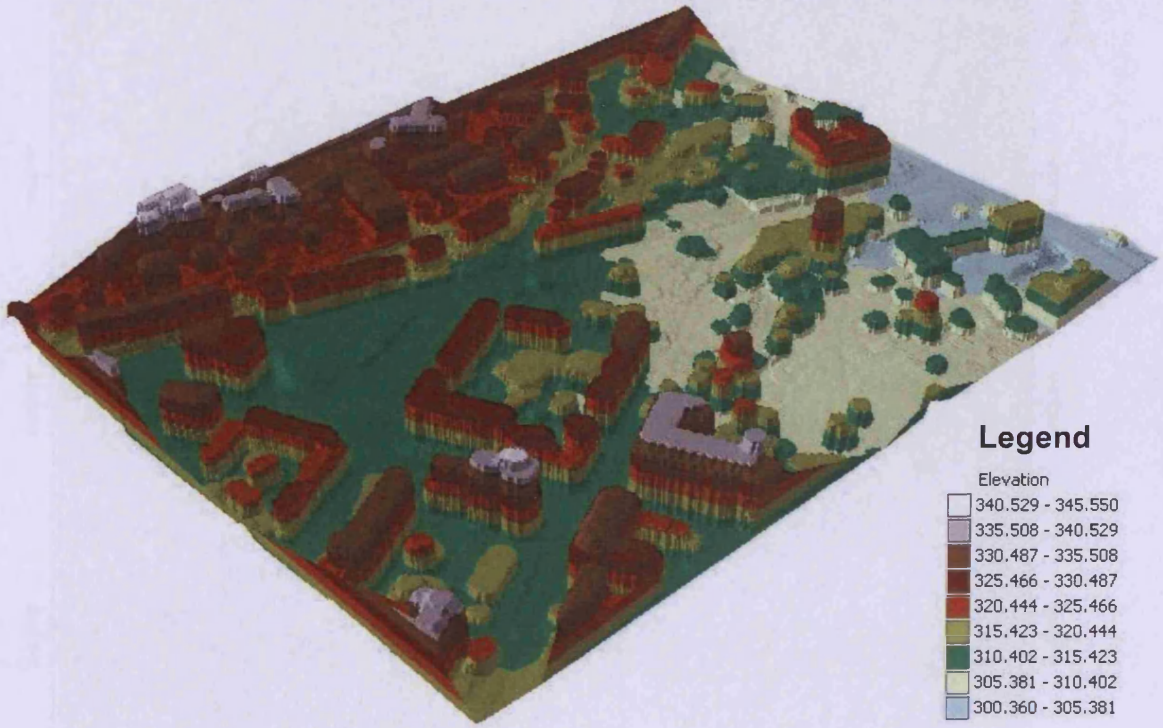


Figure 8.11 – The DSM generated from the morphologically filtered LiDAR data – Stuttgart, Site 2.

Comparing the DSMs depicted in both figures above with those in Figure 8.4 and Figure 8.5 respectively, it can be seen that the filtering process "removed" a considerable amount of LiDAR points reflected by small objects above ground. As a consequence, spatial features like buildings and trees are now much clearly standing on their own. Also, looking at the elevation range it can be seen that this was shortened in both sites, but most considerably in Site 1 with the removal of the peak in the middle of the surveyed area (Site 1: 283m-482m before, 283m-340m now; Site 2: 300m-357m before, 300m-345m now).

The first experiment carried out with morphologically filtered data used the same thresholding as for the raw data: the 70° gradient value. The maps of steep/flat regions obtained for Site 1 and Site 2 of Stuttgart from the morphologically filtered data, are depicted in Figure 8.12 and Figure 8.13 respectively. Site 1 map comprises now 1743 steep/flat polygons only (9415 before, *vd.* Figure 8.6); Site 2 map comprises 1625 polygons (8092 before, *vd.* Figure 8.7).

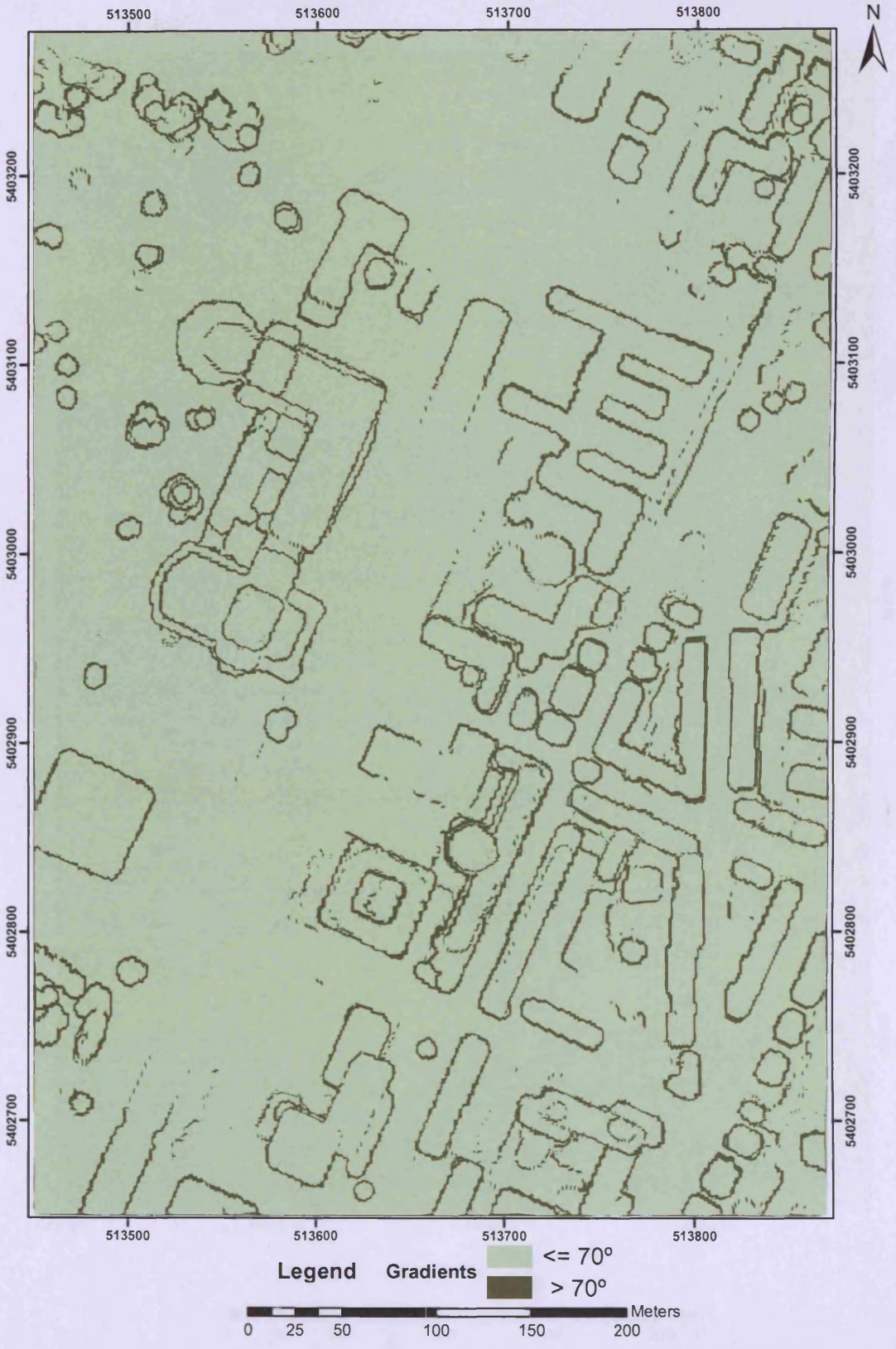


Figure 8.12 – The binary classification of the TIN facets and the generation of steep/flat polygons – Morphologically filtered data; 70° gradient thresholding.
(The Stuttgart dataset, Site 1)

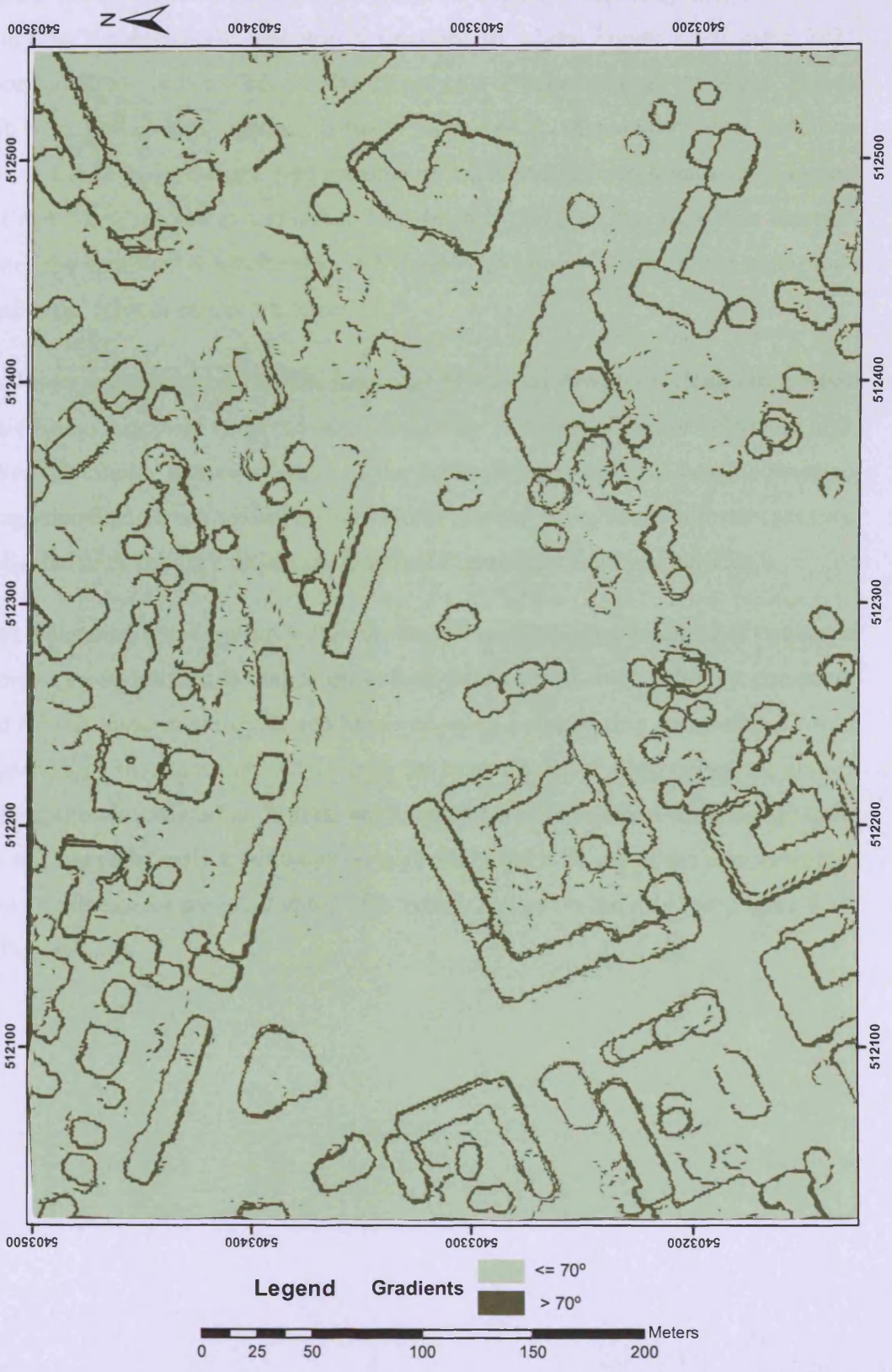


Figure 8.13 – The binary classification of the TIN facets and the generation of steep/flat polygons – Morphologically filtered data; 70° gradient thresholding.
(The Stuttgart dataset, Site 2)

Figure 8.14 and Figure 8.15 depict the results of the urban topology analysis for Site 1 and Site 2 respectively. Polygon *6* (mapped in white, Figure 8.14), with 1023 adjacent regions, is the UEB in Site 1; polygon *2* (also mapped in white, Figure 8.15), with 934 adjacent regions, is the UEB in Site 2. Glancing through these two maps, it can be seen that the performance of the algorithm was somewhat different from that when applied to unfiltered data. In fact, this time the algorithm detected most of the important urban features individually as separate containment units, such as buildings, trees or sets of trees/bushes.

In addition, it is also clear that the map edge effects are not so much an issue when the algorithm is applied to filtered data, especially if combined with a relatively high gradient threshold. However, some of the urban features are still being connected through remaining steep sliver polygons along the map edge. Some of these cases are highlighted with yellow cross on the map (*vd.* Figure 8.14 and Figure 8.15).

When generating the steep/flat polygons, the 70° gradient thresholding left out some important urban features in both sites. In fact, the chains of steep polygons enclosing these missed features are open, and hence no steep-polygon ring containment could be detected by the algorithm. As a result, the open chains of steep polygons, almost outlining the associate urban feature, are mapped but not the actual containment unit. This was not technically a failure of the algorithm, but a failure of the data example. A few of these cases are highlighted with yellow circles on the map (*vd.* Figure 8.14 and Figure 8.15).

Figure 8.14 - The urban topology analysis and the different containment units identified – Crosses indicate existence of sliver polygons; circles indicate urban features not identified as containment units. (Morphologically filtered data; 70° gradient thresholding - The Stuttgart dataset, Site 1)

Figure 8.15 - The urban topology analysis and the different containment units identified – Crosses indicate existence of sliver polygons; circles indicate urban features not identified as containment units. (Morphologically filtered data; 70° gradient thresholding - The Stuttgart dataset, Site 2)

The facts noted in the previous paragraph led to the conclusion that the 70° gradient threshold is probably too high for the filtered datasets. Thus, further experiments were accomplished by dropping the gradient threshold. Two different thresholds were considered randomly: the 60°, and the 45° gradient values.

The maps of steep/flat polygons obtained for Site 1 and Site 2 of Stuttgart using a 60° gradient threshold are depicted in Figure 8.16 and Figure 8.17 respectively. The new Site 1 map comprises 1753 polygonal regions, and the new Site 2 map comprises 1700 regions. It can be seen from both maps that by dropping the threshold some of the chains of steep polygons left open by the 70° gradient thresholding are now complete, potentially enclosing more urban features.

In order to check the consistency of the results, a third experiment using the same 60° thresholding was carried out entailing the application of the algorithm to a subset of the Site 1 dataset. Comprising a cloud of 25095 3D range points, this sub-dataset covers an approximately 145 x 200m area; it includes a building and its neighbourhood (*vd.* yellow box in Figure 8.16 and Figure 8.18).

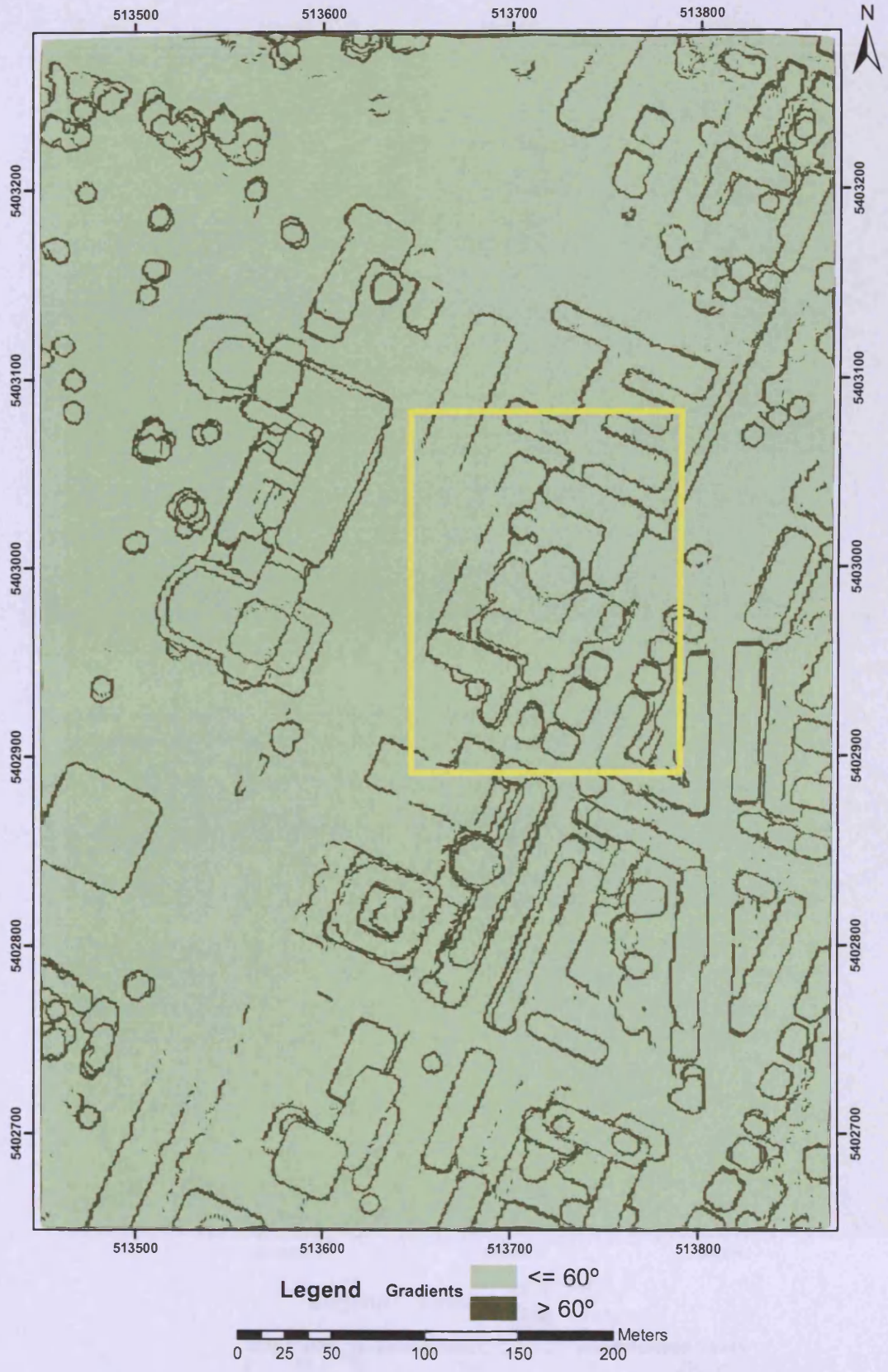


Figure 8.16 – The binary classification of the TIN facets and the generation of steep/flat polygons - Morphologically filtered data; 60° gradient thresholding. The yellow box represents a sub-dataset. (The Stuttgart dataset, Site 1)

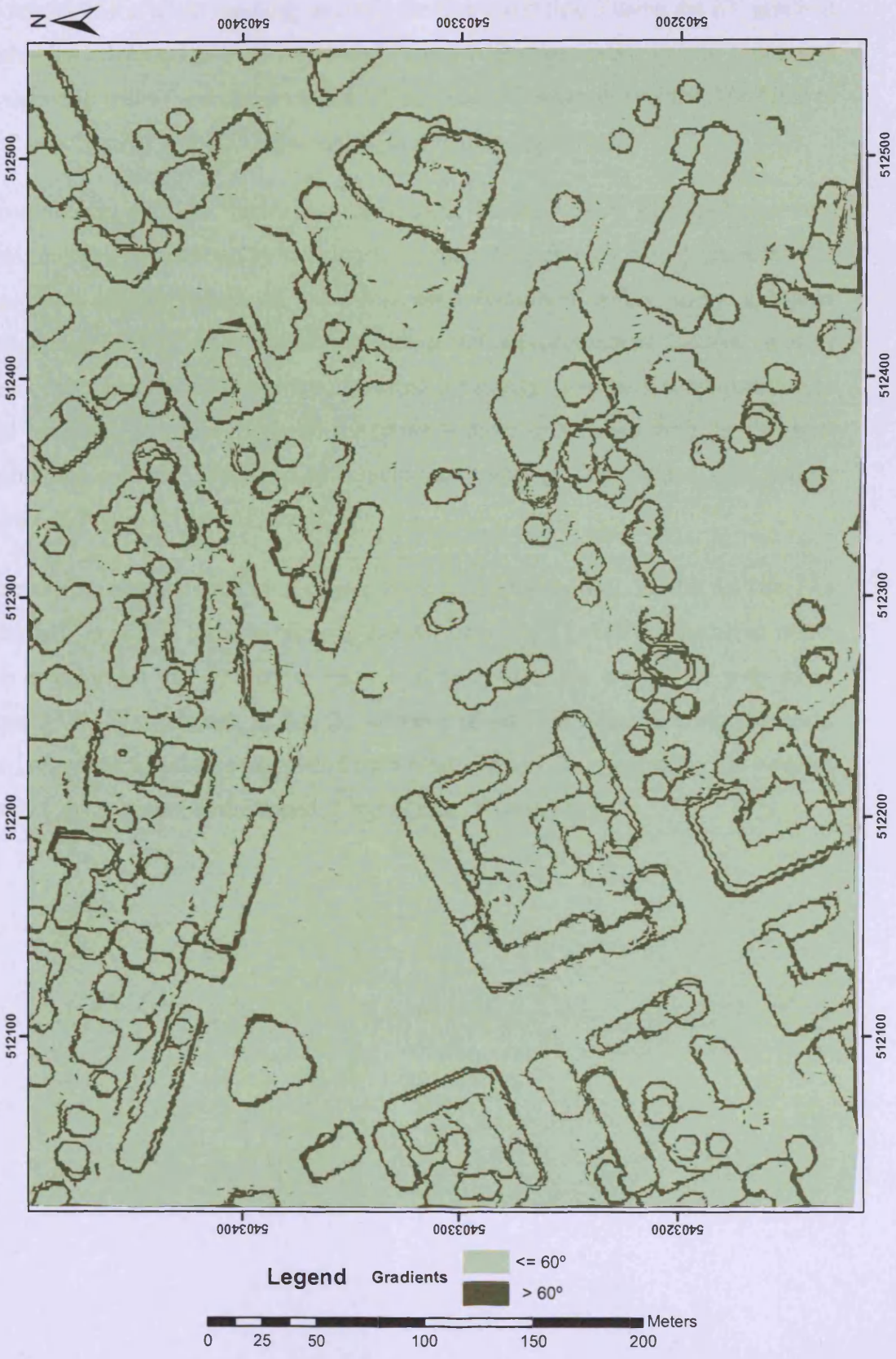


Figure 8.17 – The binary classification of the TIN facets and the generation of steep/flat polygons – Morphologically filtered data; 60° gradient thresholding. (The Stuttgart dataset, Site 2)

The results of the urban topology analysis for Site 1 and Site 2 using the 60° gradient threshold are depicted in Figure 8.18 and Figure 8.19 respectively. In Site 1, polygon *4* (mapped in white) was chosen as the UEB; it has 797 adjacent regions. The UEB in Site 2 is polygon *2* with 857 adjacencies (also mapped in white).

In comparison with the previous experiments, because more steep-polygon ring containments were detected by the algorithm, more containment units were identified as separate entities. Most of these had been identified before using a higher thresholding, but they are larger now including new adjacent spatial features; in other cases, new containment units were identified separately, generally corresponding to small buildings and trees. Some of the urban features missed out with the previous thresholding and now identified as containment units, are highlighted with yellow circles (*vd.* Figure 8.18 and Figure 8.19).

However, the extent of the map edge effects is greater as well. As far as Site 1 is concerned, any urban features lying on the map edge were linked and gathered in the same containment unit (*vd.* containment unit mapped in red around the map edge, Figure 8.18). With regards to Site 2, the same effect is noticeable along the north edge (*vd.* containment unit mapped in light blue, Figure 8.19), and along the western edge (*vd.* containment unit mapped in light green, Figure 8.19).

Figure 8.18 - The urban topology analysis and the different containment units identified – Crosses indicate existence of sliver polygons; circles indicate urban features missed out with the previous thresholding. (Morphologically filtered data; 60° gradient thresholding - The Stuttgart dataset, Site 1)

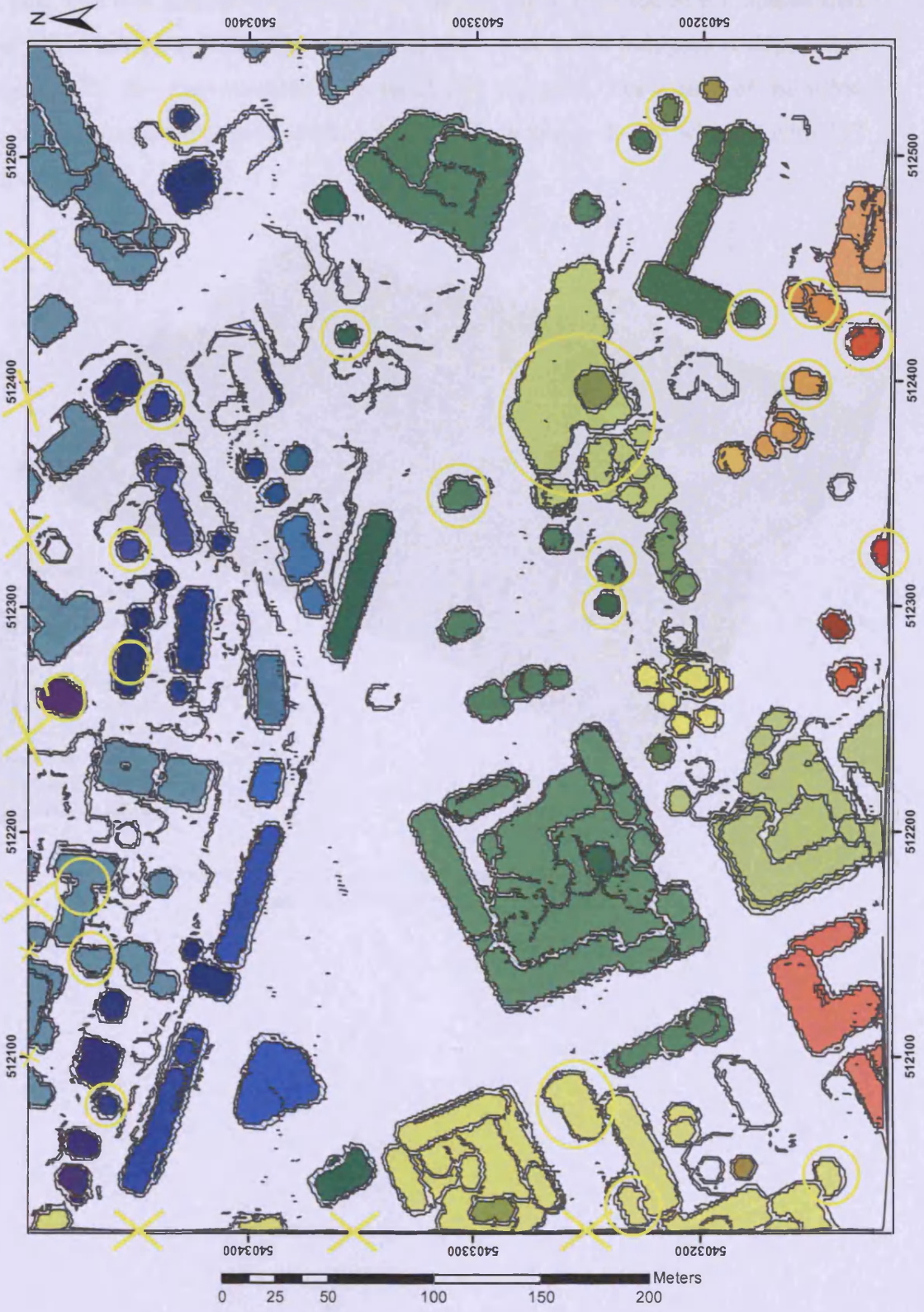


Figure 8.19 - The urban topology analysis and the different containment units identified – Crosses indicate existence of sliver polygons; circles indicate urban features missed out with the previous thresholding. (Morphologically filtered data; 60° gradient thresholding - The Stuttgart dataset, Site 2)

A new TIN was generated across the sub-dataset taken from the Site 1 filtered data (*vd.* DSM in Figure 8.20). The respective map of steep/flat polygons is depicted in Figure 8.21; this map comprises a total of 209 polygons. The results of the urban scene analysis are shown in Figure 8.22 (UEB: polygon *2* – in white – with 107 adjacencies).

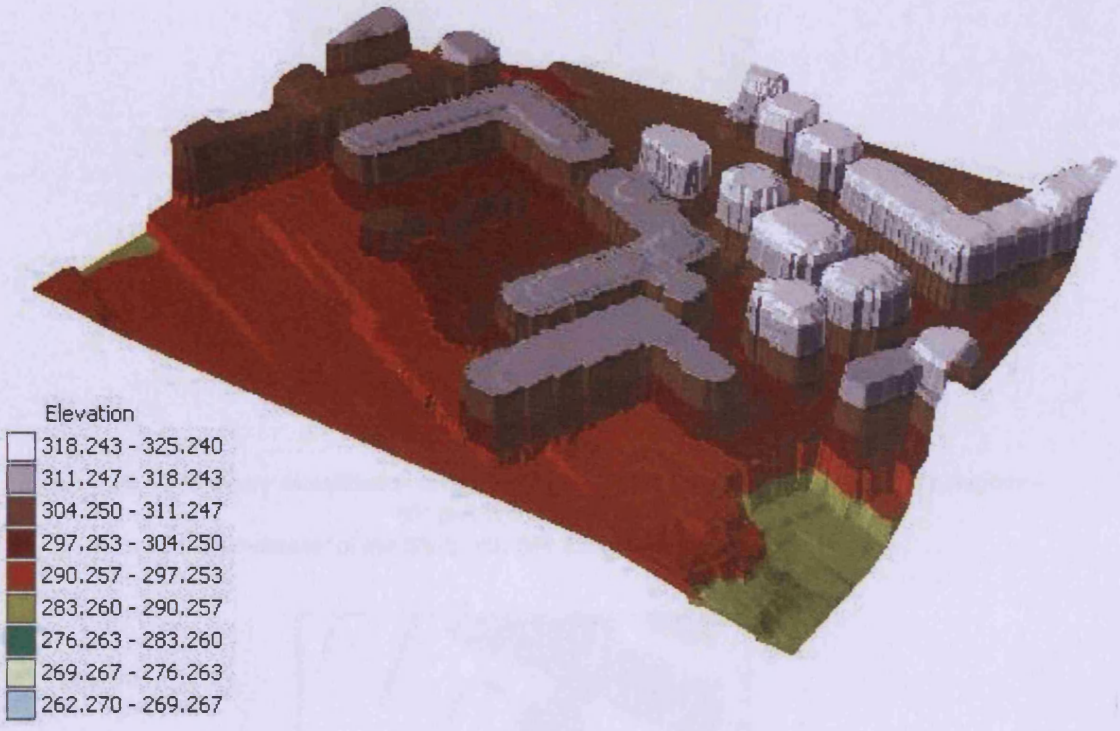


Figure 8.20 – The DSM generated from a sub-dataset of the Stuttgart's Site 1 morphologically filtered data.

Figure 8.21 – The binary classification of the TIN facets and the generation of steep/flat polygons – 60° gradient thresholding.
(A sub-dataset of the Stuttgart's Site 1 morphologically filtered data)

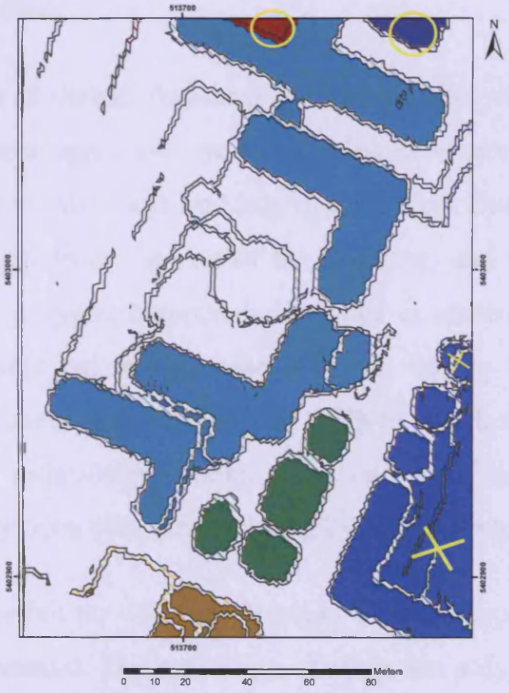


Figure 8.22 - The urban topology analysis and the different containment units identified– 60° gradient thresholding.
(A sub-dataset of the Stuttgart's Site 1 morphologically filtered data)

When comparing and contrasting the results pictured above in Figure 8.22 and those pictured back in Figure 8.18, it can be said that the results obtained with the two datasets are basically the same, though there are a couple of small details worth of examination.

The algorithm identified the same containment units in this sub-dataset, as done before when the complete dataset was used. By examining Figure 8.18, it can be seen that the block in the southeast corner of the yellow box was now identified as a single containment unit (*vd.* southeast corner of Figure 8.22). In fact, in the yellow box in Figure 8.18, the back yard - in white - of the buildings in green, and the small building mapped in light blue (on the east edge of the yellow box) were gathered in a single containment unit, mapped in navy (*vd.* yellow crosses in the southeast of Figure 8.22); this was because of the map edge effect which linked together all buildings in the southeast corner in a single containment unit (Figure 8.22). In addition, when the whole dataset was used, the ground polygon corresponding to the back yard was part of the UEB; but, when considering this sub-dataset, this back yard was split from the rest of the UEB by the edge map, and was included in the containment unit identified.

As far as the north part of the sub-dataset is concerned, one polygon in dark blue and another in dark red are mapped (*vd.* yellow circles in Figure 8.22). The dark blue polygon is actually part of the large building in light blue. But again, because of the map edge, it was cut off from the rest of the building, and turned into a separate containment unit. The polygon mapped in dark red is another consequence of the edge effects. What happened in this case was that during the adjacencies graph traversal the search process moved from the UEB to the dark red polygon via the universe polygon. An independent containment unit was therefore identified, and hence a different colour from blue was assigned by the algorithm.

The last experiment carried out with the Stuttgart datasets entailed the consideration of the 45° gradient threshold. The new maps of steep/flat polygons obtained for Site 1 (consisting of 2222 steep/flat polygons) and Site 2 (consisting of 2065 steep/flat polygons) of Stuttgart are depicted in Figure 8.23 and Figure 8.24 respectively. It can be seen from both maps that by considering a lower threshold more steep-polygon chains, which had been left open by the previously used thresholds, are now

complete, defining new containment units. However, it is also true that with such a low threshold more noise was inserted into the maps of steep/flat polygons.

The results of the urban topology analysis for Site 1 and Site 2 are depicted in Figure 8.25 and Figure 8.26 respectively. In Site 1, the UEB is polygon *4* (mapped in white), with 712 adjacent polygons; the UEB in Site 2 is polygon *12* (also mapped in white), which has 866 adjacencies.

Apart from one small feature or another (corresponding to a small building or a tree), the algorithm identified basically the same containment units as before (especially with the 60° gradient thresholding). In other instances, the previously identified containment units expanded now to integrate nearby spatial relations of containment newly detected by the algorithm.

As noted above, given the low threshold used in this experiment, the number of meaningless steep/flat polygons (in terms of the urban scene) is greater. Nevertheless, the extent of the map edge effects is practically the same as that in the experiment with a 60° gradient thresholding. In fact, the containment units identified before around the map edge are also identified with the 45° gradient threshold: in Site 1, one unit along the western edge was identified; another unit was identified all around the northern, eastern and southern edges (*vd.* Figure 8.25); in Site 2, one unit was identified along the western edge; another unit was identified along the northern and part of the eastern edges (*vd.* Figure 8.26).

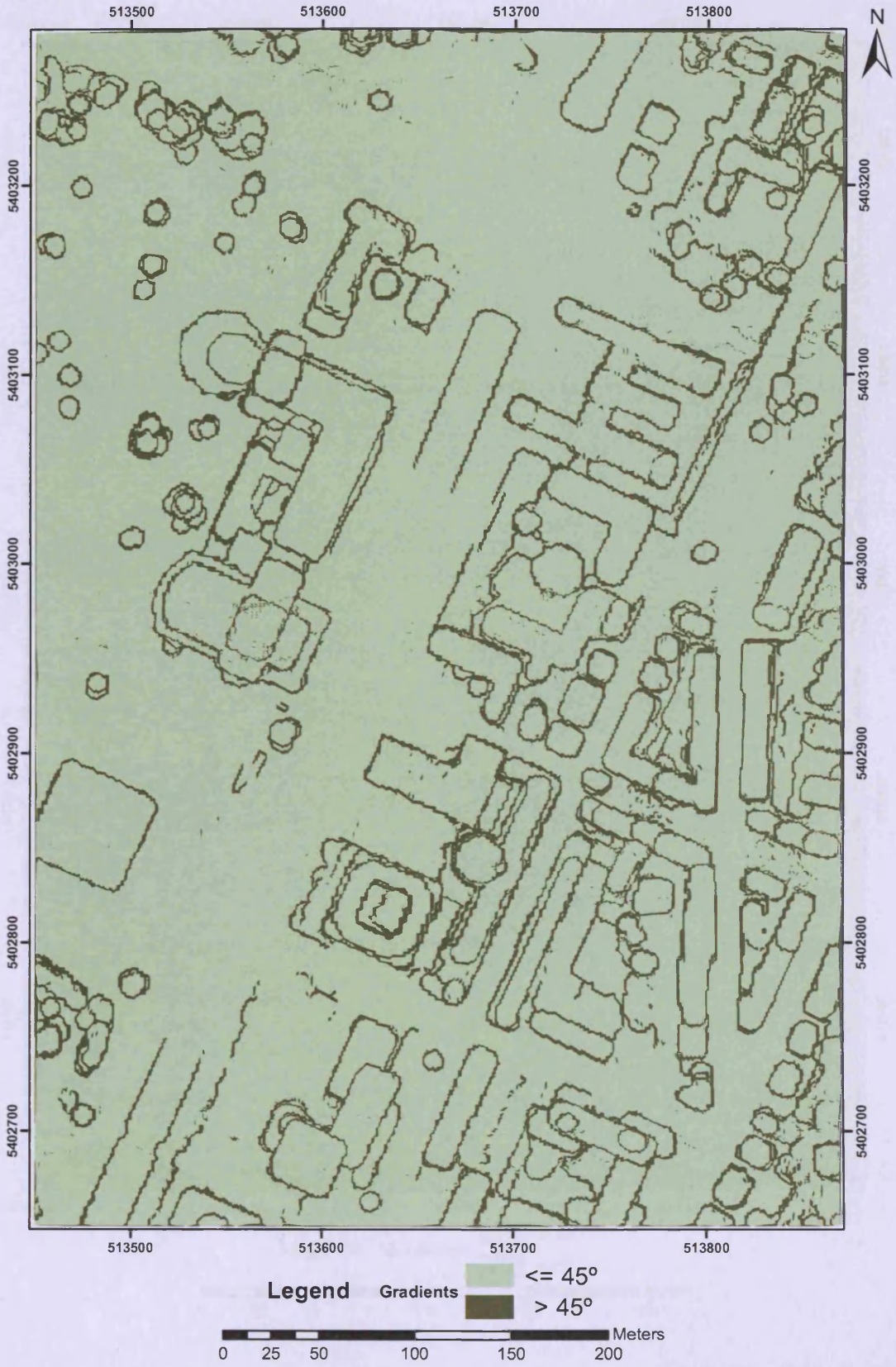


Figure 8.23 – The binary classification of the TIN facets and the generation of steep/flat polygons – Morphologically filtered data; 45° gradient thresholding.
(The Stuttgart dataset, Site 1)

Figure 8.24 – The binary classification of the TIN facets and the generation of steep/flat polygons – Morphologically filtered data; 45° gradient thresholding. (The Stuttgart dataset, Site 2)

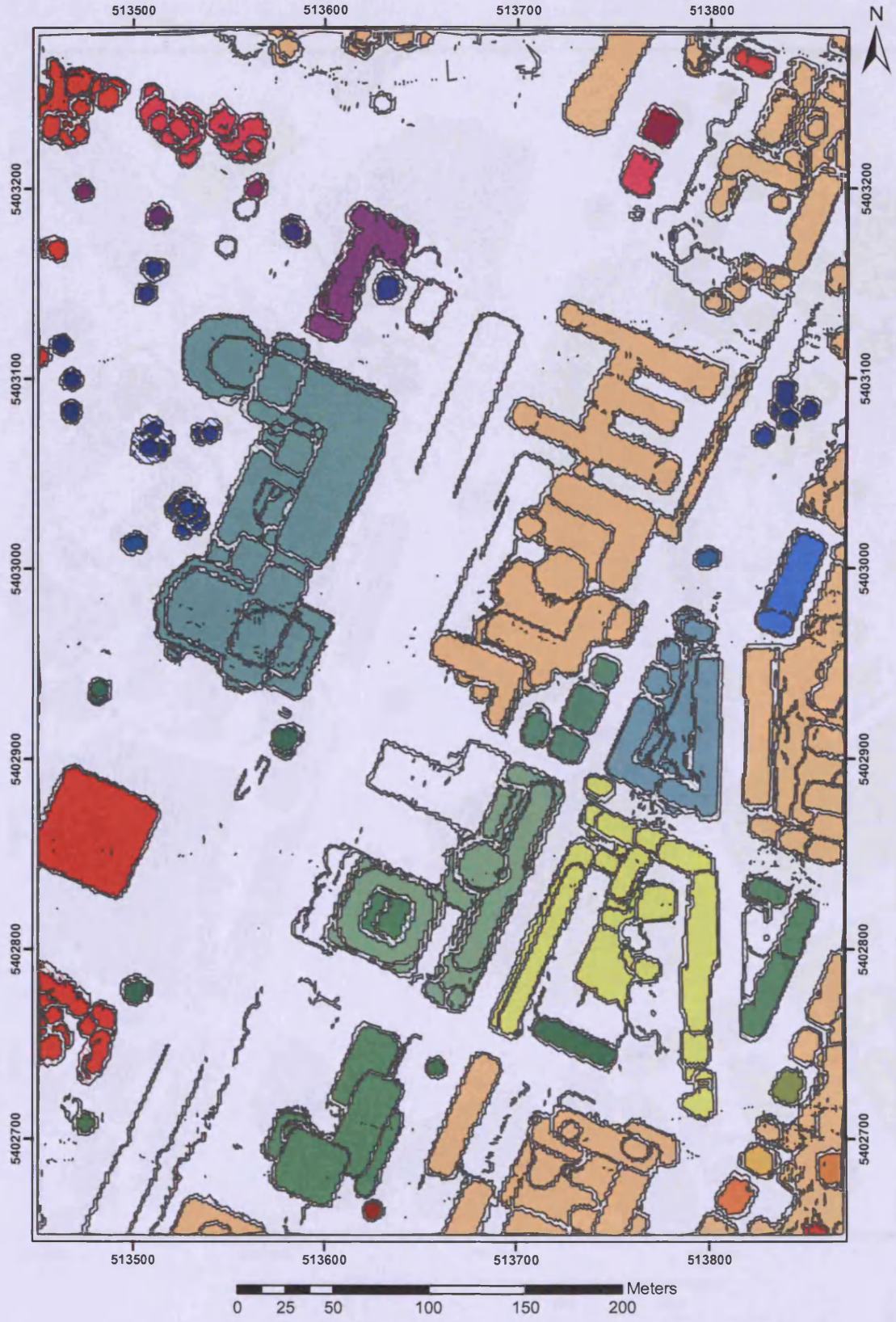


Figure 8.25 - The urban topology analysis and the different containment units identified – Morphologically filtered data; 45° gradient thresholding. (The Stuttgart dataset, Site 1)

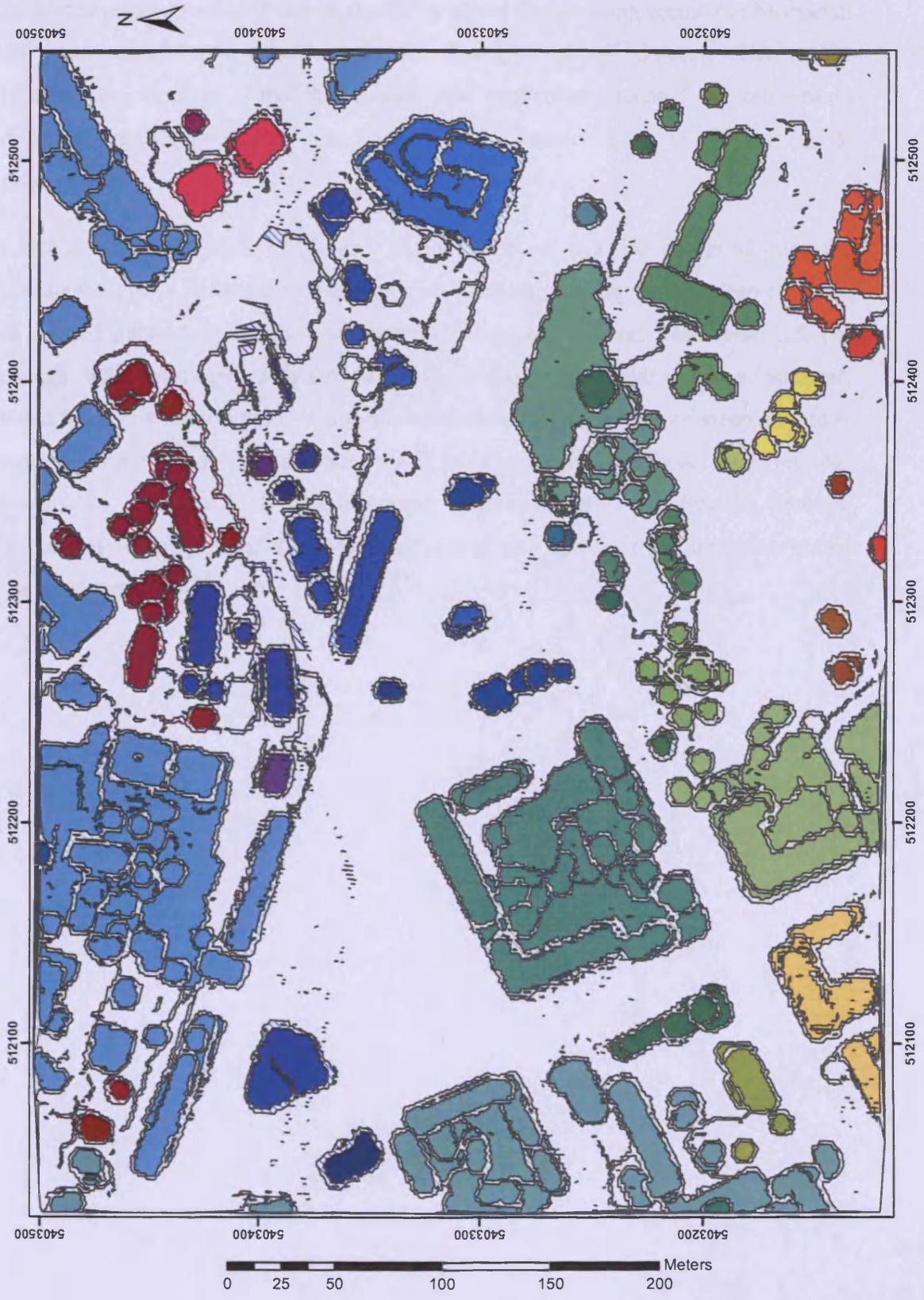


Figure 8.26 - The urban topology analysis and the different containment units identified – Morphologically filtered data; 45° gradient thresholding. (The Stuttgart dataset, Site 2)

Given the results presented above, the 60° gradient thresholding seemed to be overall the most successful thresholding for the Stuttgart LiDAR datasets. The results obtained for the Site 1 and Site 2 with that particular threshold are once more depicted in Figure 8.27 and Figure 8.28, respectively, overlaying aerial photography[11].

Glancing through Figure 8.27 and Figure 8.28, it can be observed that the containment units detected by the algorithm correspond to the main urban features, or sets of urban features. A few small buildings and several trees were left out though. With this regard, two different reasons should be noted: in some instances, those features were left out because the associate outlining chains of steep polygons were left open by the thresholding used; in other instances, those were left out because the associate LiDAR points were "removed" during the filtering process. These reasons suggest that that fact is not related to a failure of the algorithm but of the input data.

[11] Provided by Norbert Haala, University of Stuttgart (Germany).

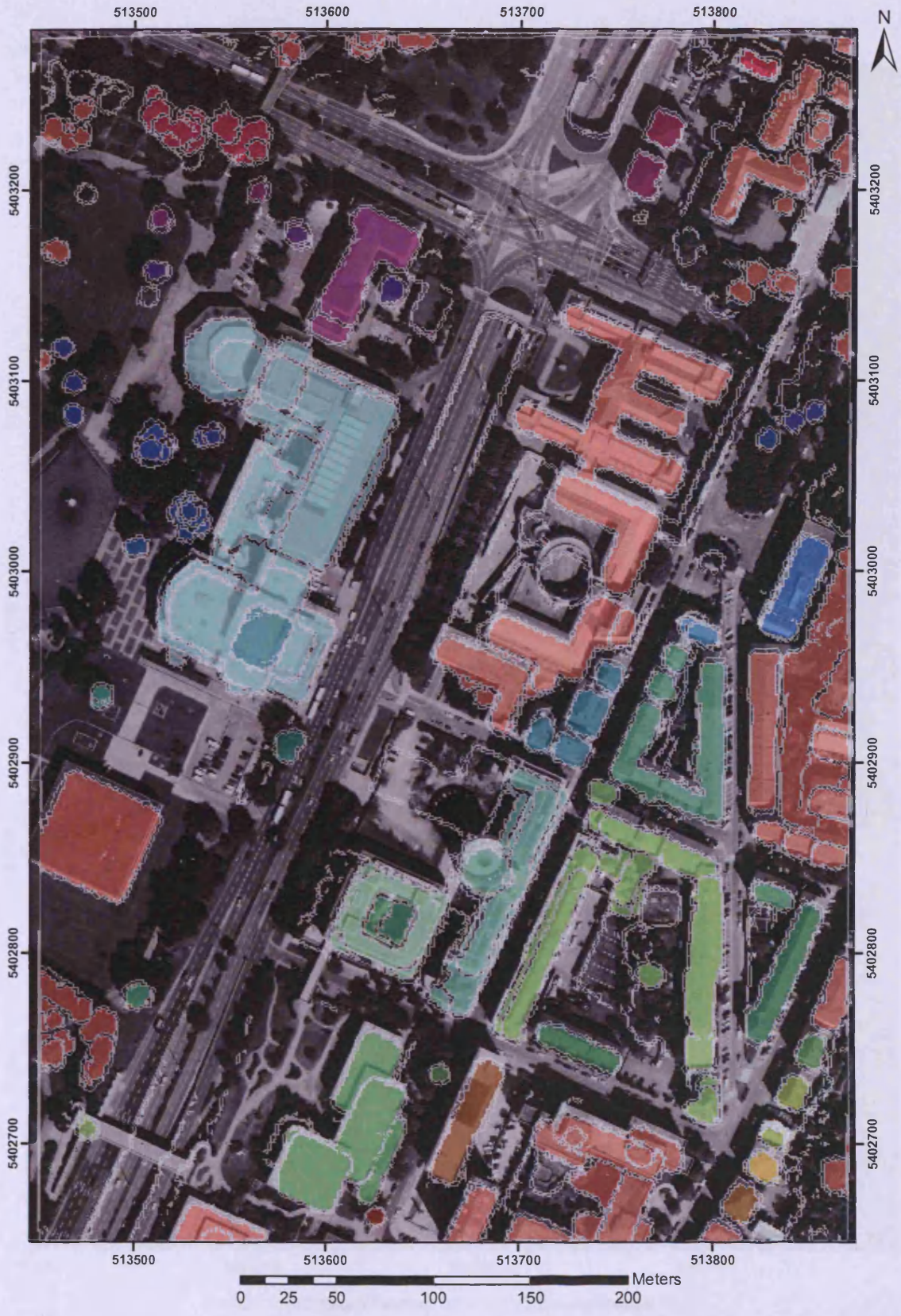


Figure 8.27 – The urban topology analysis: the identified containment units overlaying aerial photography.
(The Stuttgart's Site 1; morphologically filtered data; 60° gradient thresholding)

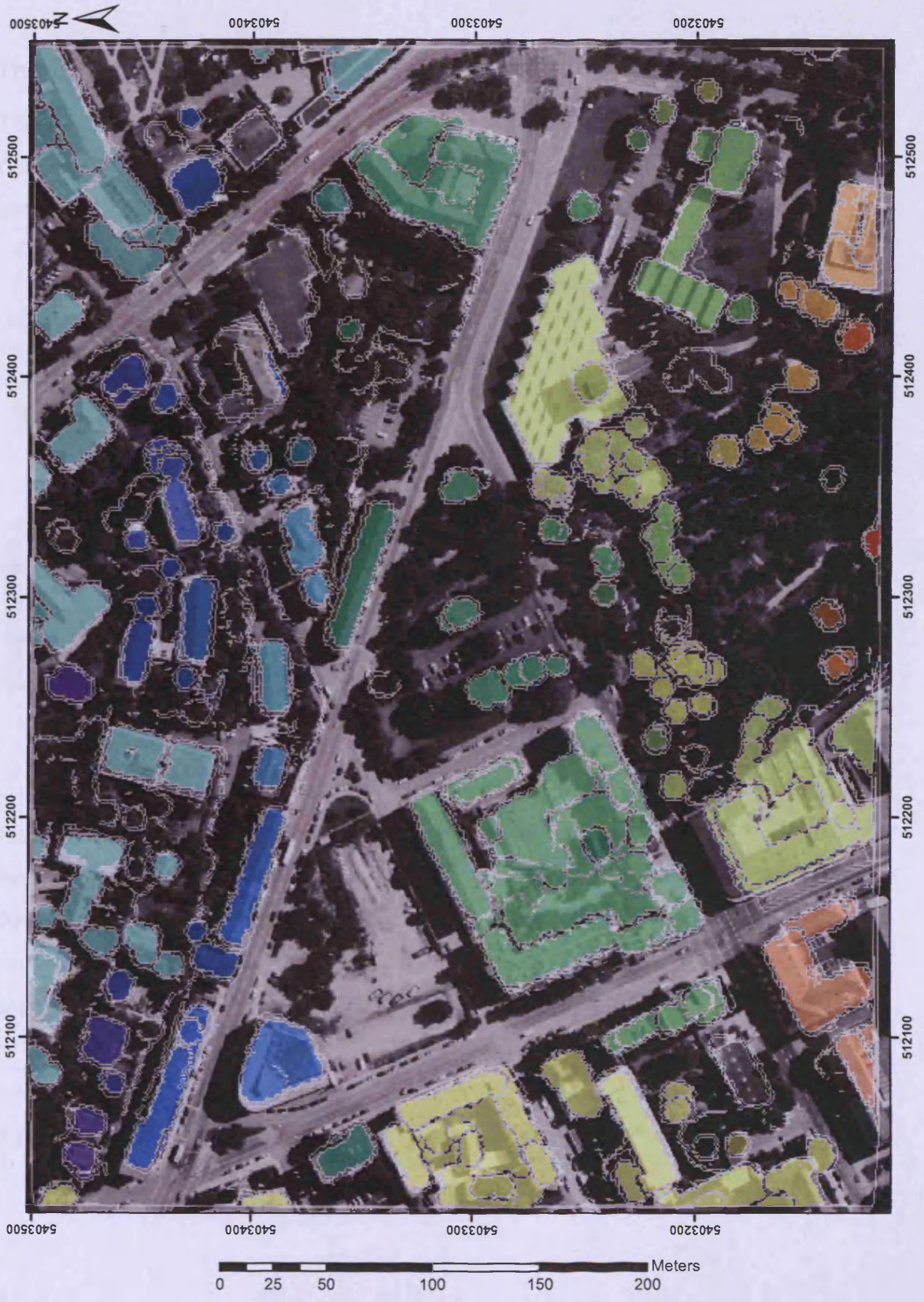


Figure 8.28 – The urban topology analysis: the identified containment units overlaying aerial photography.
(The Stuttgart's Site 2; morphologically filtered data; 60° gradient thresholding)

8.2 Experiments with photogrammetric data

8.2.1 The Barton-Bendish dataset

The dataset covering the Barton-Bendish area was pre-processed as described in Chapter 5, section 5.4.2. Topological information was brought in to the dataset by generating a TIN. The DSM obtained is depicted in Figure 8.29 below.

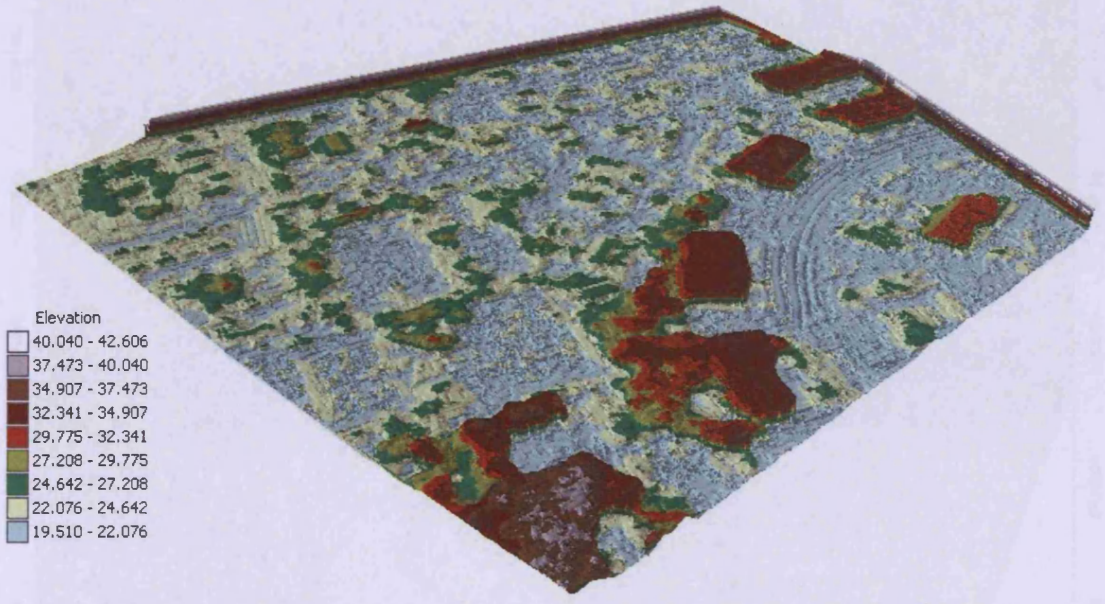


Figure 8.29 – The DSM generated from the photogrammetric point set of Barton-Bendish.

Although this dataset has 1m point spacing, preliminary tests showed that gradient thresholds of 70° or even 60° are high for these data. What happens with such a high thresholding is that the largest man-made structures in the area are not sufficiently outlined by the steep-polygon chains. This is principally because the point cloud is aligned to a regular grid (the reason why is also awkward to decipher the urban scene by a simple visual inspection of the DSM above). Thus, the first experiment carried out used a 45° gradient threshold. The map of steep/flat polygons obtained is depicted in Figure 8.30; it comprises 10370 steep/flat polygons.

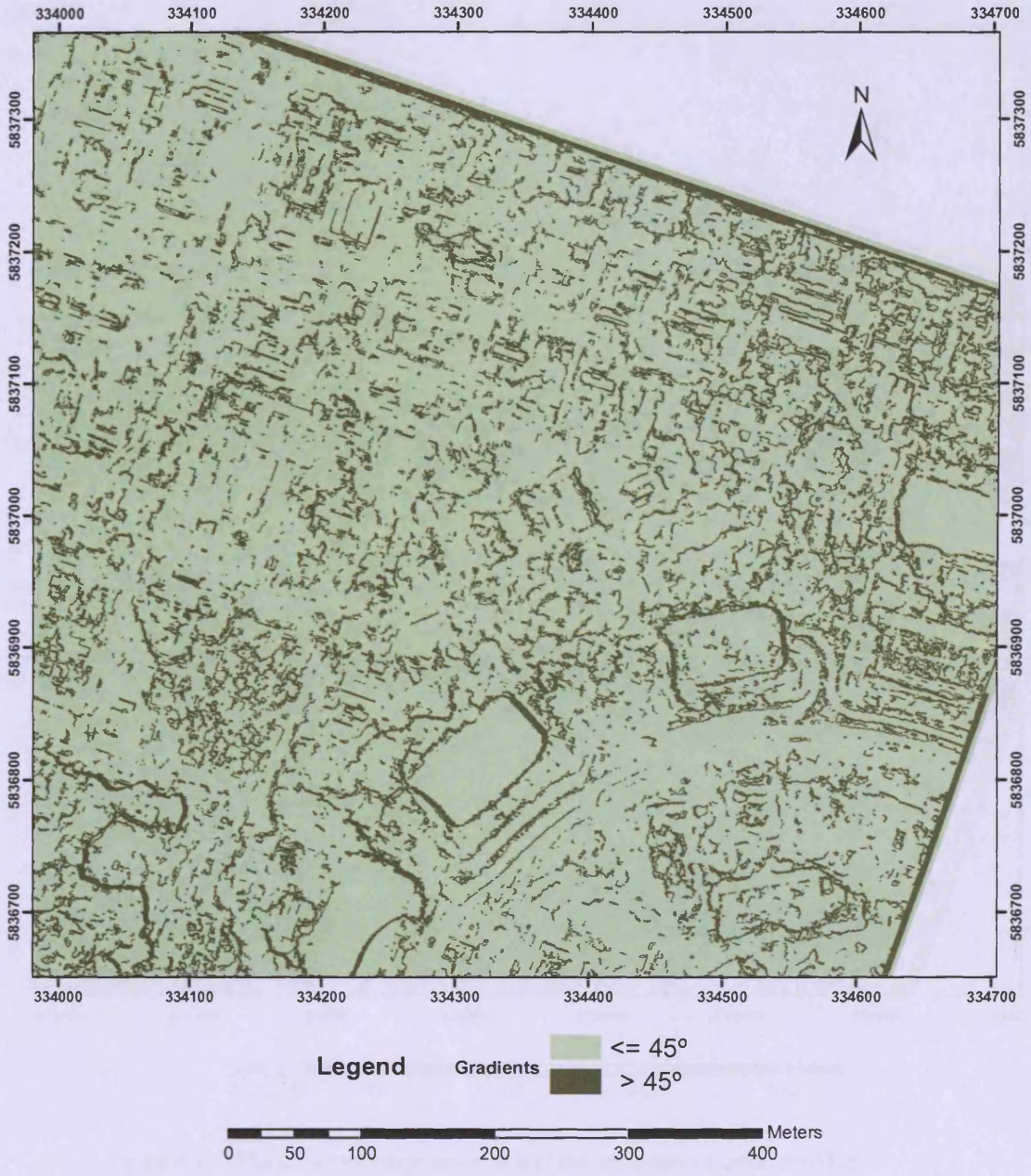


Figure 8.30 – The binary classification of the TIN facets and the generation of steep/flat polygons - using the 45° gradient threshold (The Barton-Bendish dataset).

It can be seen from the map in Figure 8.30 that even the 45° gradient threshold is not appropriate for this kind of data. For example, the largest buildings in the southeast of the dataset are still not well outlined by the steep polygons.

Nevertheless, the analysis of the urban topology was carried out for this case. The UEB chosen for the analysis of the adjacencies graph was polygon *3* with 7863 adjacent polygons. The results of the analysis are shown in Figure 8.31 below (the UEB, polygon *3*, is mapped in white).

Figure 8.31 - The urban topology analysis and the containment units identified – using the 45° gradient threshold (The Barton-Bendish dataset).

As expected, the vast majority of the urban features, including the four large buildings in the southeast area, ended up merged with the UEB. A couple of small containment units were identified (some of which are highlighted in yellow on the map in Figure 8.31), but they do not really have any meaning in terms of the urban scene, *i. e.* do not correspond to either or both buildings and vegetation. Furthermore, all the containment units identified also ended up gathered in a single structure to which the red colour was assigned.

A lower threshold was considered and a second experiment was carried out with these data. The 30° gradient value was chosen. The binary classification of the TIN facets using this threshold resulted in the generation of 15097 steep/flat polygons (*vd.* Figure 8.32). The major man-made structures are now slightly better outlined by chains of steep polygons; however, the steep-polygon chain shaping one of the large buildings (in the south) was still left open by this threshold.

The results of the urban topology analysis of these data are shown in Figure 8.33. Some of the major buildings in the southeast area (all in red) were correctly identified as containment units; the woodland in the southwest corner was also correctly identified as a containment unit; other small units were detected across the dataset but these are in general meaningless in terms of the urban scene. Moreover, once again all the units of containment identified are linked across the whole dataset constituting a single structure (mapped in red).

Two main reasons are believed to explain the unsatisfactory results obtained with the photogrammetric dataset of Barton-Bendish. The fact that the photogrammetric points are aligned to a regular grid resulted in the generation of a DSM not accurate enough for the purposes of this study. In addition, 1m point spacing is not especially high resolution for photogrammetric data. Thus, technically, this represents not a failure of the analysis method of the urban topology, but a failure of the test data.

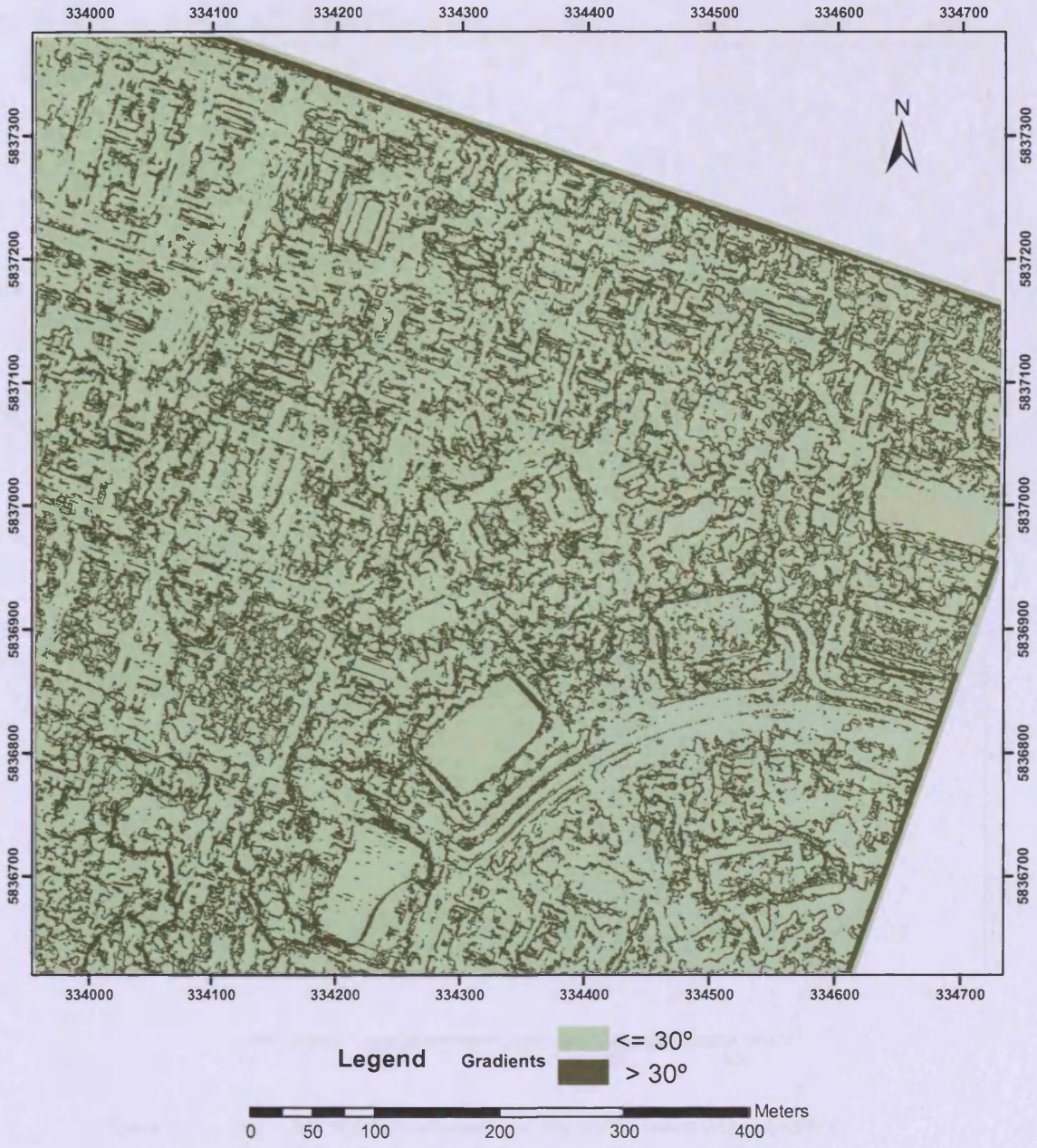


Figure 8.32 - The binary classification of the TIN facets and the generation of steep/flat polygons - using the 30° gradient threshold (The Barton-Bendish dataset).

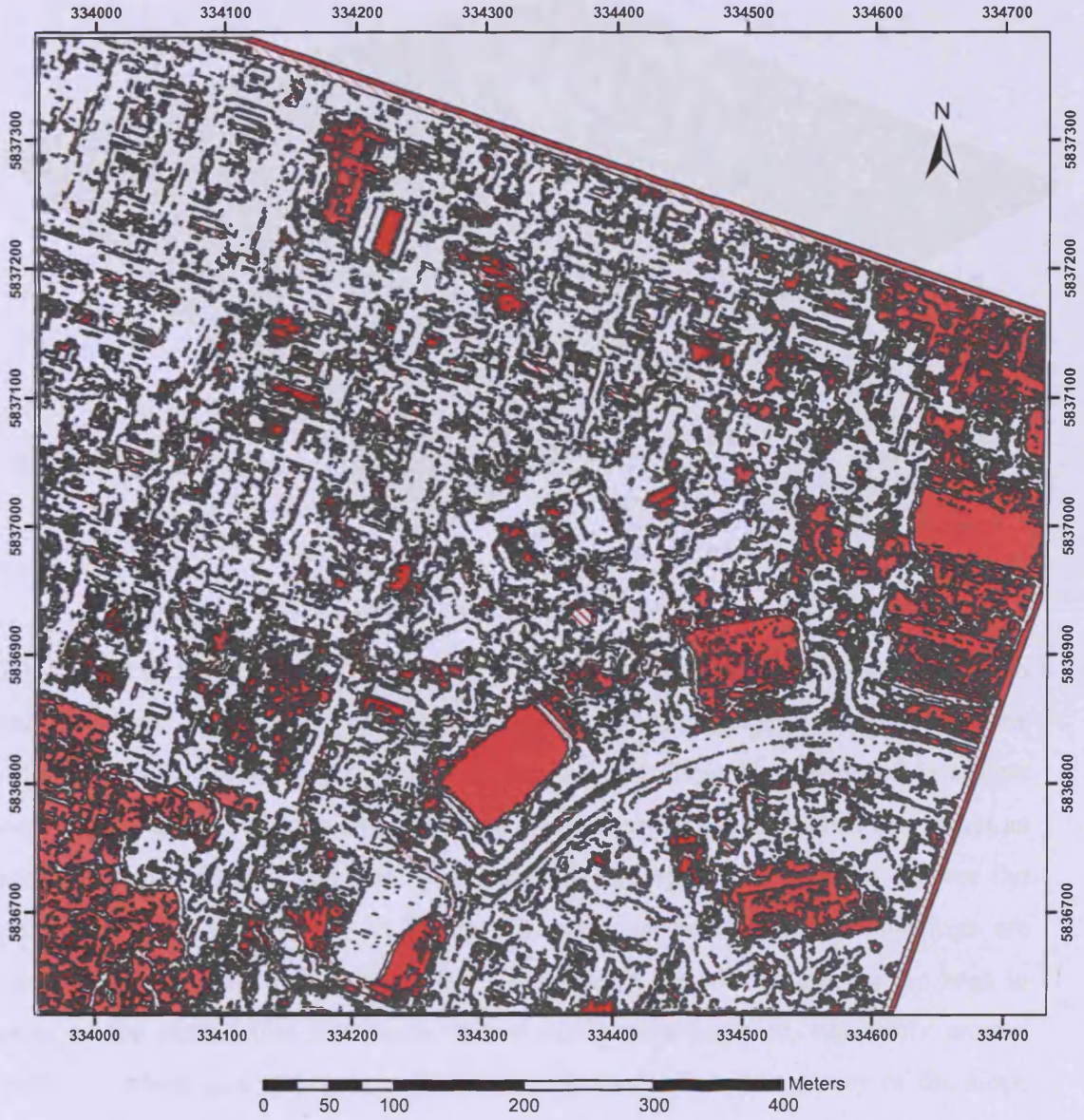


Figure 8.33 - The urban topology analysis and the containment units identified – using the 30° gradient threshold (The Barton-Bendish dataset).

8.2.2 The Heerbrugg dataset

A TIN was generated throughout the cloud of photogrammetric points of this dataset, as described in Chapter 5, section 5.4.2. The DSM obtained is depicted in Figure 8.34. Comparing this DSM with all the DSMs generated from the other datasets, it can be seen that this is not meaningful enough in terms of the urban scene. At a first glance, no other features are detectable on the Earth surface apart from one or two alignments of objects, corresponding to main roads.

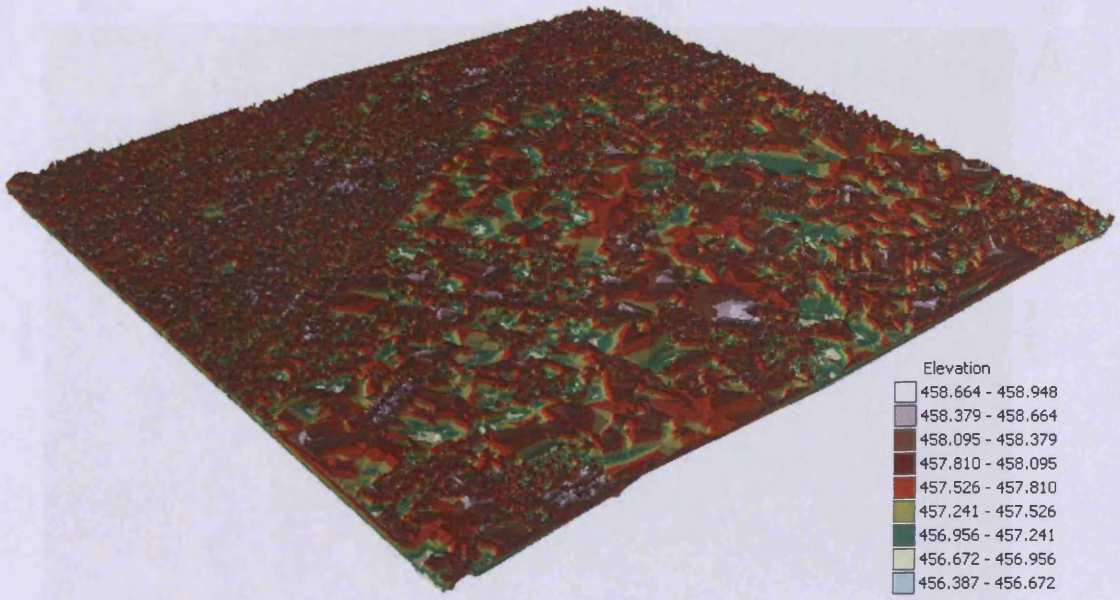


Figure 8.34 - The DSM generated from the photogrammetric point set of Heerbrugg.

Figure 8.35 shows part of the surveyed area: some of the building polygons (Kokkas and Dowman, 2006) are displayed on top of aerial photography; an extract of the original photogrammetric point cloud is overlaid over these two layers of data. One can observe that the cloud of points is not particularly dense; however, this is not so much an issue. In fact, the important aspect is the spread of locations where the photogrammetric points lie. As far as this study is concerned, the buildings are considered under sampled – just a few points along the roof ridges can be seen in most of the cases. Also the terrain is not sufficiently sampled, especially around buildings where ground points are most needed for an accurate survey of the slope discontinuities. For these reasons, no further experimentation was carried out towards the analysis of the urban topology.

The attempt made with the photogrammetric dataset of Heerbrugg offers clear evidence that high point density is not so much a requirement for the algorithm to be applied. Most importantly is the spread of locations where the points lie, and how sufficiently sampled both terrain and geographic features (especially buildings) are.

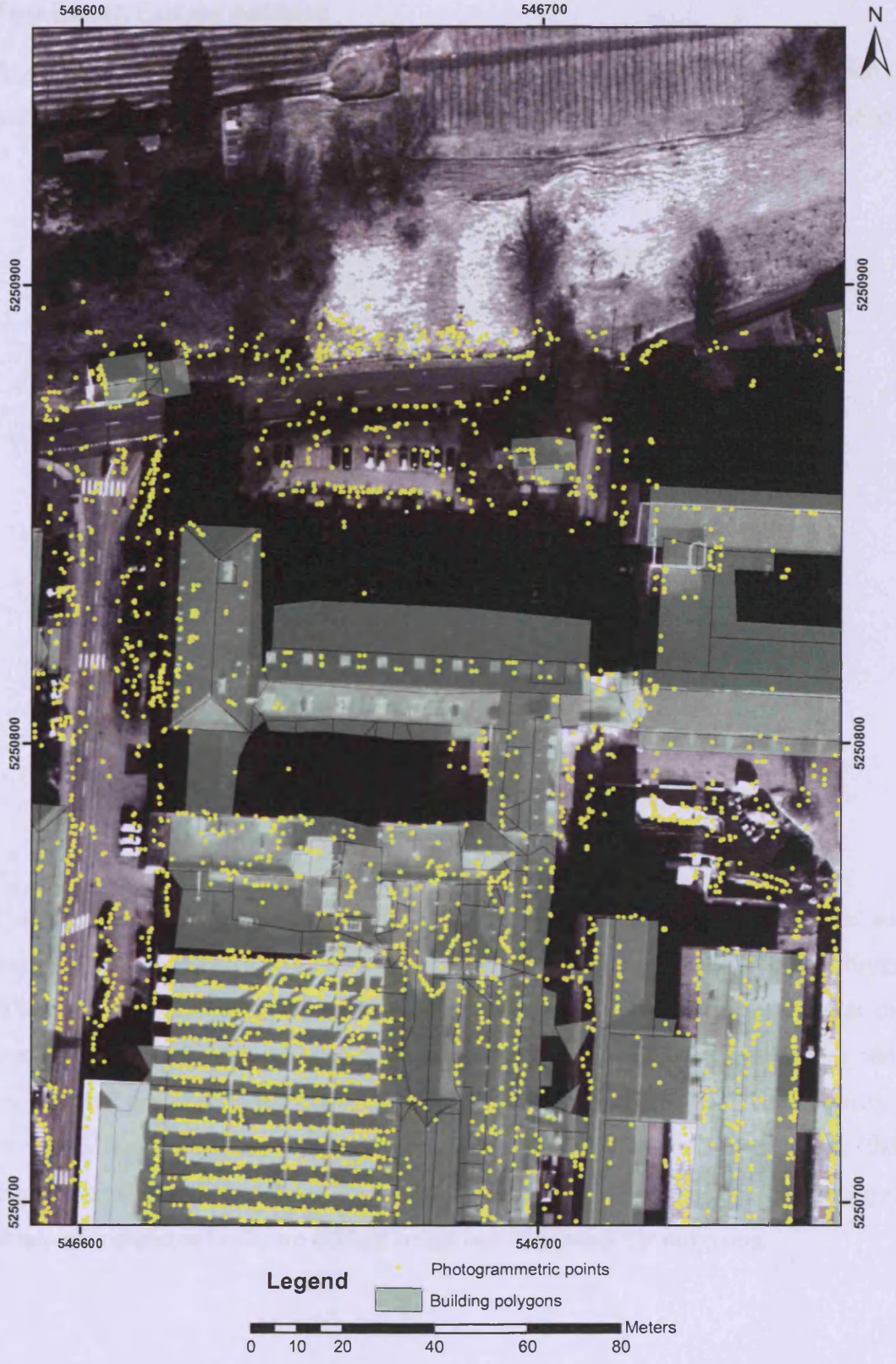


Figure 8.35 – Extract from the Heerbrugg photogrammetric data: the urban features, like buildings, are under sampled for the purposes of this research.

8.2.3 The Santa Lucija dataset

As accomplished with the other datasets, a TIN was generated across the cloud of photogrammetric points of the Santa Lucija dataset. Figure 8.36 illustrates the DSM.

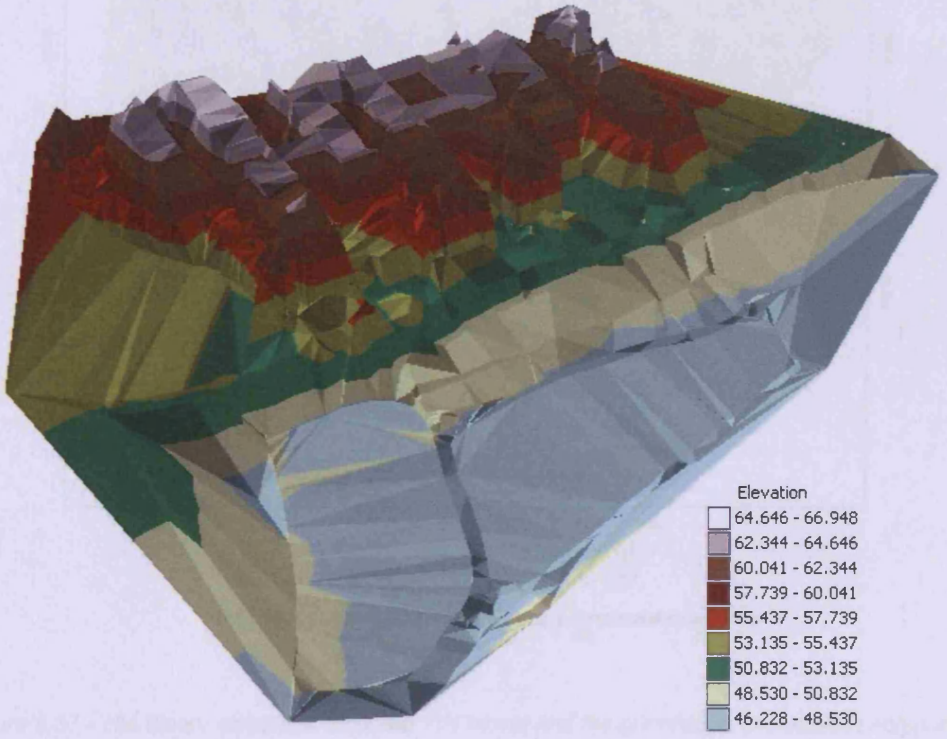


Figure 8.36 - The DSM generated from the photogrammetric point set of Santa Lucija.

Covering a 53270m^2 area and comprising 1343 points, this is a low density point set. Even so, the generated DSM is more meaningful than, for instance, the Heerbrugg DSM. In fact, the location of the main man-made structures can be observed at the top of Figure 8.36, aligned in the northwest-southeast direction. Preliminary tests showed that gradient thresholds of 60° or even 45° are high for this dataset, resulting in urban features poorly shaped by the steep-polygon chains. Hence, the first experiment carried out used a 30° gradient threshold. The map of steep/flat polygons obtained is depicted in Figure 8.37; it comprises 153 steep/flat polygons.

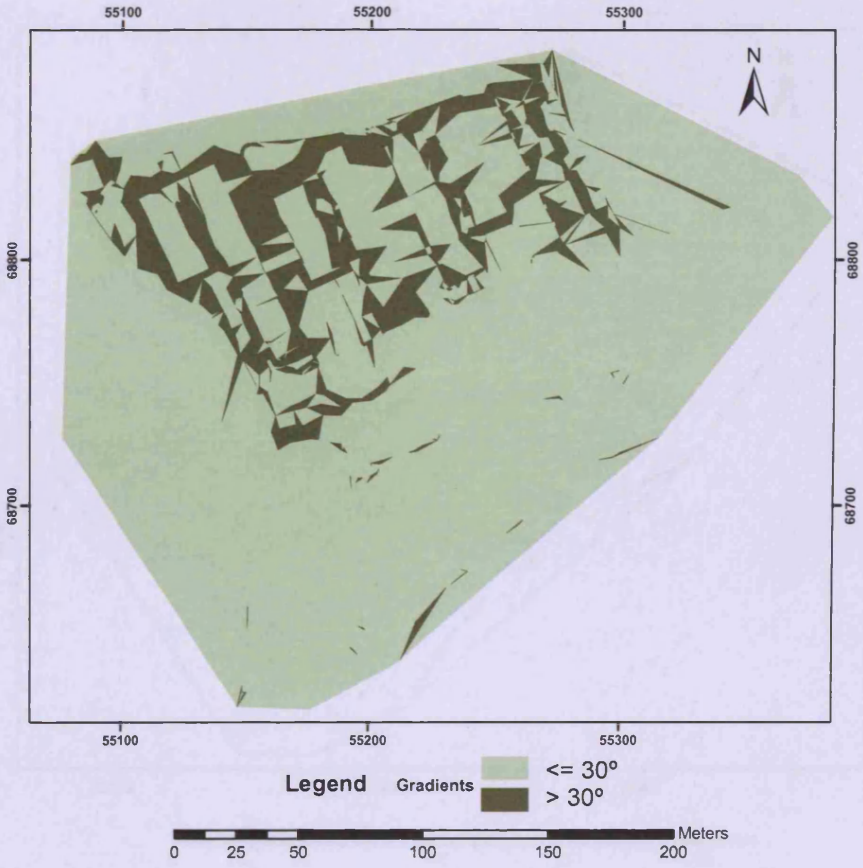


Figure 8.37 - The binary classification of the TIN facets and the generation of steep/flat polygons - using the 30° gradient threshold (The Santa Lucija dataset).

Using the 30° gradient threshold, the analysis of the urban topology was carried out by choosing polygon *6* as the UEB; this polygon has 32 adjacent polygons. The results of the urban topology analysis are shown in Figure 8.38 (the UEB, polygon *6*, is mapped in white).

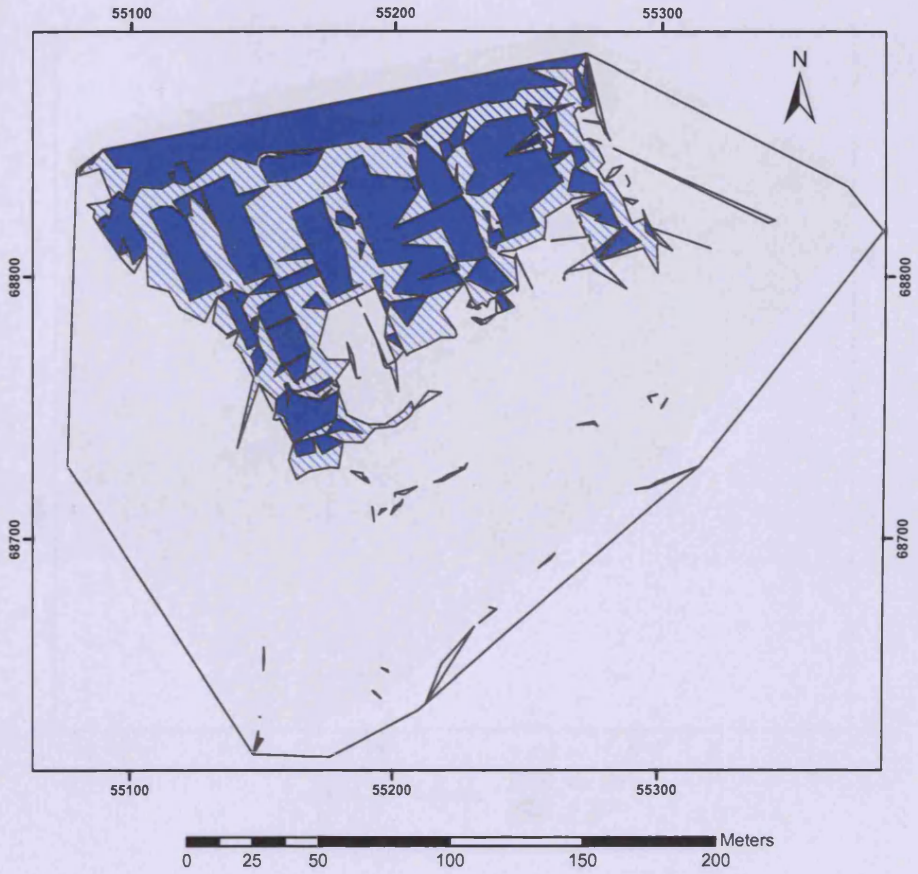


Figure 8.38 - The urban topology analysis and the containment units identified – using the 30° gradient threshold (The Santa Lucija dataset).

Figure 8.38 shows that all the relationships of containment detected by the algorithm were assembled in a single unit (mapped in tones of blue).

An interesting aspect of the results depicted in Figure 8.38 is that all the flat polygons (with the exception of the large polygon at the top of the figure) - mapped in solid blue - correspond roughly to the building footprints. In turn, the surrounding steep polygons - in hashed pattern - represent the enclosing road network. The reason why the road network was not separated from the actual steep areas is due to the lack of photogrammetric points sampling the ground.

Another experiment was carried with this dataset taking into account a lower gradient threshold of 20°. The revised map of steep/flat polygonal regions, comprising 172 polygons, is depicted in Figure 8.39.

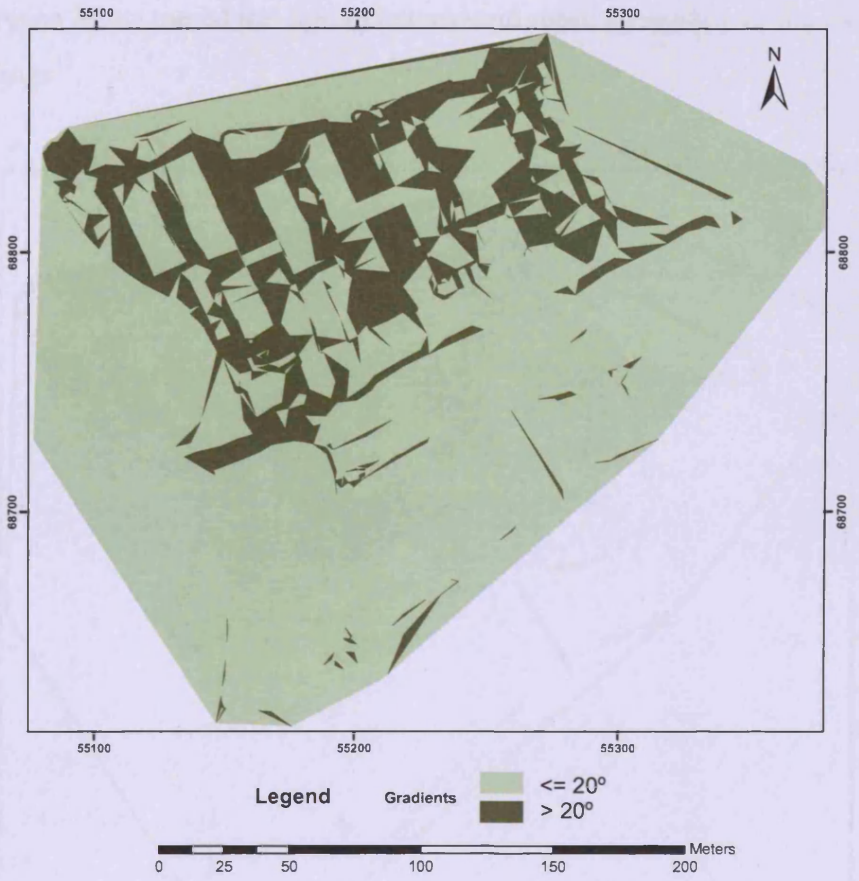


Figure 8.39 - The binary classification of the TIN facets and the generation of steep/flat polygons - using the 20° gradient threshold (The Santa Lucija dataset).

The results of the urban topology analysis using the 20° gradient threshold are depicted in Figure 8.40. The UEB selected is polygon *7* (mapped in white), with 51 adjacent polygons.

The results obtained are generally the same as those previously obtained with a 30° gradient thresholding. The relationships of containment detected were all assembled again in a single unit (mapped in tones of purple).

Because a lower threshold was used, the noise was relatively increased in the map of steep/flat polygons (*e. g.* more steep-polygon islands were generated). On the other hand, the flat polygons - mapped in solid purple - (again, with the exception of the

large polygon at the top of the figure) correspond more accurately to the footprint of the buildings[12].

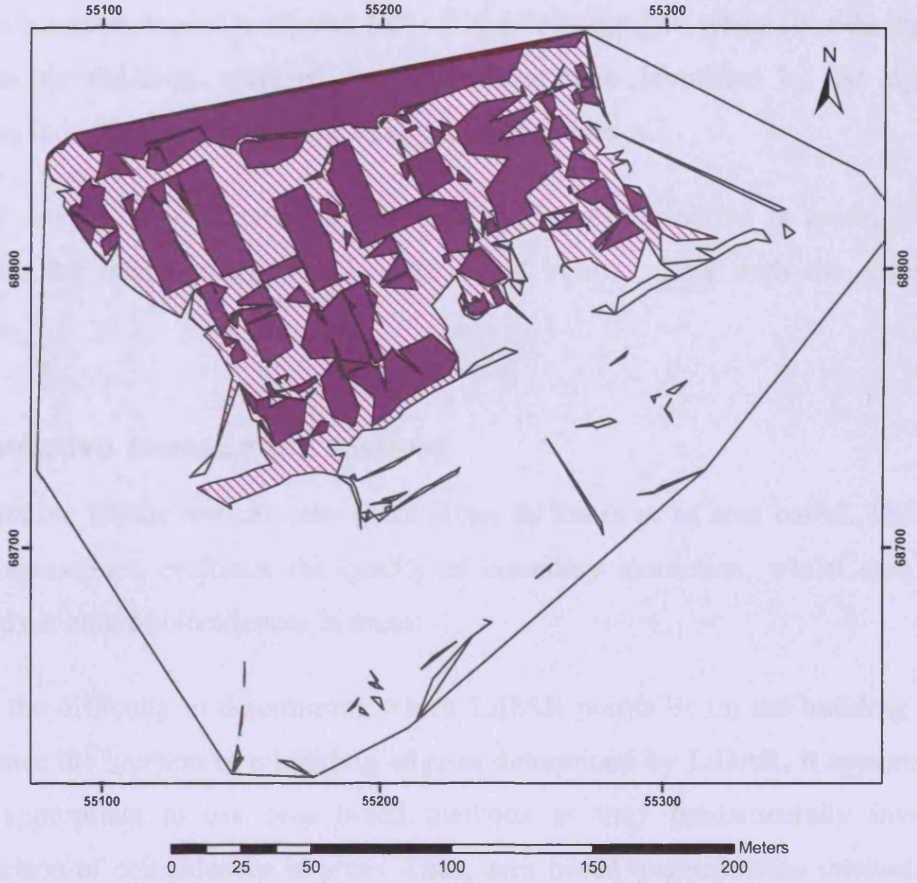


Figure 8.40 - The urban topology analysis and the different containment units identified – using the 20° gradient threshold (The Santa Lucija dataset).

With such a low density photogrammetric point set, it was still possible to obtain relatively meaningful results, when compared to the other photogrammetric datasets. The facts that, in both experiments, the relationships of containment detected were all assembled in a single unit, and the fact that the road network was not separated from the actual steep areas, are due to the lack of photogrammetric points on the ground. In turn, the buildings are sampled enough for our purposes.

[12] To actually show this, the results should be presented overlaying aerial photography; however, it was not possible to make these data available in time to be included in this thesis.

8.3 Statistical evaluation of key results

A visual inspection of the results obtained suggested that the algorithm performed best with morphologically filtered LiDAR data of Stuttgart, when considering a 60° gradient thresholding: most of the containment units identified by the algorithm relate to individual urban features like buildings and trees.

In this section, a quantitative assessment of the results above is made. For this purpose, the reference data used is described below, along with the assessment criteria.

8.3.1 Quantitative assessment method

Quantitative assessment can take place either as linear or as area based. The linear based assessment evaluates the quality of boundary extraction, whilst area based methods compare coincidences in areas.

Given the difficulty in determining which LiDAR points lie on the building edges, and hence the location of a building edge as determined by LiDAR, it appears to be more appropriate to use area based methods as they fundamentally involve a comparison of coincidences in areas. Thus, area based quantification methods were used in this thesis to compare reference feature polygons and containment-unit areas detected by the algorithm.

The quality measures used in this thesis were those defined by Shufelt (1999). Although his metrics were primarily defined and used for the quality assessment of building and tree detection, their concepts were borrowed and applied in this thesis to evaluate to what extent the detected containment units relate to spatial features.

For the definition of those metrics, Shufelt focused on the development and application of unbiased metrics for scene level comparative analysis. These are based on the comparison of measurements to an object and its background in both the reference and classification space. In computing Shufelt's metrics, the reference scene model is taken to be the foreground whilst the algorithm's scene model is taken to be the background and all metrics computed are said to be "in the reference space" - these are equivalent to "producer accuracies" that are generated using error

matrices. In the second instance, the algorithm's scene model is taken to be the foreground whilst the reference scene model is taken to be the background, with the recomputed metrics considered as being "in the object space" - these are equivalent to the "user accuracies" in error matrices (Shufelt, 1999). Thus, four possible categories of classification arise, namely:

- **True positive (TP):** both the reference scene model and the algorithm's scene model classify a pixel as being part of an object;
- **True negative (TN):** both the reference scene model and the algorithm's scene model classify a pixel as being part of the background;
- **False positive (FP):** the algorithm's scene model classifies a pixel as part of an object, but the reference scene model classifies the same pixel as being part of the background;
- **False negative (FN):** the algorithm's scene model classifies a pixel as part of the background, but the reference scene model classifies the same pixel as being part of an object.

To evaluate performance, the values TP, TN, FP and FN are determined and the following metrics computed (Shufelt, 1999):

- **Object detection percentage (ODP):** $\frac{100TP}{TP+FN}$;
- **Branching factor (BF):** $\frac{FP}{TP}$;
- **Quality percentage (QP):** $\frac{100TP}{TP+FN+FP}$.

The ODP measures the fraction of the reference polygons that are correctly classified as a particular object (*e.g.* buildings), and is often treated as a measure of "object detection performance". If FN = 0, this measure would be 100%, and for this to occur all objects would have to be correctly classified by the algorithm. By examining the ODP formula, it can also be concluded that this percentage decreases with an increase in the value of FN (Shufelt, 1999).

The BF is a measure of how the algorithm classifies background as an object. This factor will be zero, the best possible branching factor, if the algorithm correctly identifies the background everywhere; similarly, it will be one if the algorithm misclassifies every background pixel as part of an object. The BF can be treated as a measure of "delineation performance" (Shufelt, 1999).

The QP combines aspects of both ODP and BF measures to summarise the algorithm performance; it measures the absolute quality of the scene model produced by the algorithm. A 100% value will be obtained if the system correctly identifies all objects without missing any object part (*i.e.* FN = 0), and without including any background as part of an object (*i.e.* FP = 0) (Shufelt, 1999).

These metrics were computed overall for the results obtained for both test sites considering two reference scene models. Table 8.1 summarises the computation of these metrics both in the reference and in the classification spaces. The diagram depicted in Figure 8.41 illustrates two intersecting polygons which may represent a reference polygon and a classified object polygon.

	Reference space	Classification space
True positive (TP)	$A \cap B$	$A \cap B$
False positive (FP)	$A' \cap B$	$A \cap B'$
False negative (FN)	$A \cap B'$	$A' \cap B$

Table 8.1 – Computation of Shufelt metrics.

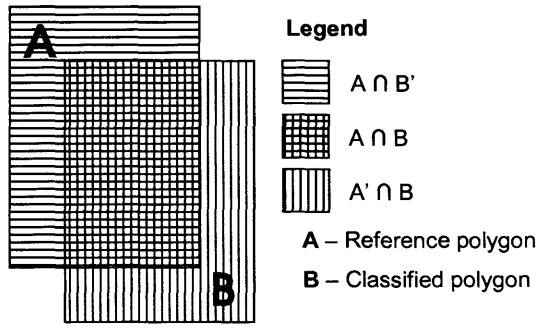


Figure 8.41 – Illustration of the computation of Shufelt metrics. (After Ayugi, 2006)

The reference and the classified polygons were overlaid in ArcMap in order to generate the necessary data for carrying out the assessment of the classification of the containment units. For the overlay operation, the PAT info files (Chapter 2, section 2.6) were accessed for the extraction of polygons, their identifiers and their area sizes. Also extracted were the overlay polygon data, which described the individual

polygons that were overlaid, and the resulting overlay areas. This information is summarised in Tables 8.2, 8.4, 8.6 and 8.8.

In assessing the quality of the detection of containment units, the area statistic is used. Primarily, an evaluation is made based on a quantitative assessment as defined above.

8.3.2 Reference data for quality assessment

Quantitative accuracy assessment is possible where reference data is available. In general, reference data can be either generated from a different and more accurate method, or generated from a more accurate data source.

The algorithm is designed to identify containment units, each of which comprises a set of enclosures between steep and flat polygons. Although the algorithm was not directly designed to identify urban features, when applied to spatial data the units of containment identified potentially correspond to urban features. Thus, the reference scene models created comprise polygons representing urban features: buildings and trees.

To generate the urban-feature polygons, two different data sources were used: the aerial image available; and the original LiDAR point cloud, colour coded according to the point heights. Urban-feature polygons were digitized, using on-screen digitization in ArcMap, in order to provide reference polygons to determine how closely the containment units identified by the algorithm match the reference scene models of the same areas.

Defining what a reference polygon should comprise was not a straightforward task. In this study it was assumed that a containment unit potentially relates to a spatial feature. Thus, urban features were primarily considered in the generation of reference polygons. The digitizing procedure for the reference polygons aimed to follow discontinuities on the terrain in terms of gradients. All the areas left out are implicitly part of the background, *i.e.* the UEB from the algorithm's perspective.

The reference scene models for Site 1 and Site 2 of Stuttgart obtained by digitization of the aerial image are shown in Figure 8.42 and Figure 8.43 respectively.

Issues faced during digitization of the aerial image included:

- The shadows next to vegetation and buildings. In particular, the digitization of building polygons was difficult especially where vegetation was overhanging the roof tops. In all these cases, the actual extent of the vegetation and building polygons could only be approximated.
- In some areas it was not possible to distinguish building edges due to the low contrast in the image.
- The digital image was supposedly ortho-rectified which would have meant the removal of distortion due to relief resulting in the position of objects being projected orthogonally onto their planimetric location. This means that the sides of the building walls would not be visible, which however is not the case, since building sides are clearly visible, suggesting that the image was not ortho-rectified[13]. Building reference polygons digitization was governed by the building roof outlines, thus creating a potential distortion between roof outlines as obtained by processing LiDAR data and roof outlines as digitized.

[13] Further information on the meta-data for the ortho-rectification had been unsuccessfully sought from the University of Stuttgart (Ayugi, 2006).

Figure 8.42 – The reference scene model for Site1 of Stuttgart, obtained by digitization of the aerial image.

Figure 8.43 - The reference scene model for Site2 of Stuttgart, obtained by digitization of the aerial image.

The reference scene models for Site 1 and Site 2 of Stuttgart obtained by digitization of the original LiDAR point cloud, are shown in Figure 8.44 and Figure 8.45 respectively.

Issues faced during digitization of the LiDAR point cloud:

- No feature outlines are available. In particular, as far as building outlines are concerned, it was very difficult to determine which LiDAR points lie on the edge of a building, and hence the location of a building edge as determined by LiDAR.
- Where the gradient discontinuities on the terrain occur, there is a high likelihood of a spatial feature being present. The reference polygon boundaries were generated by linking the LiDAR points of the same height class.

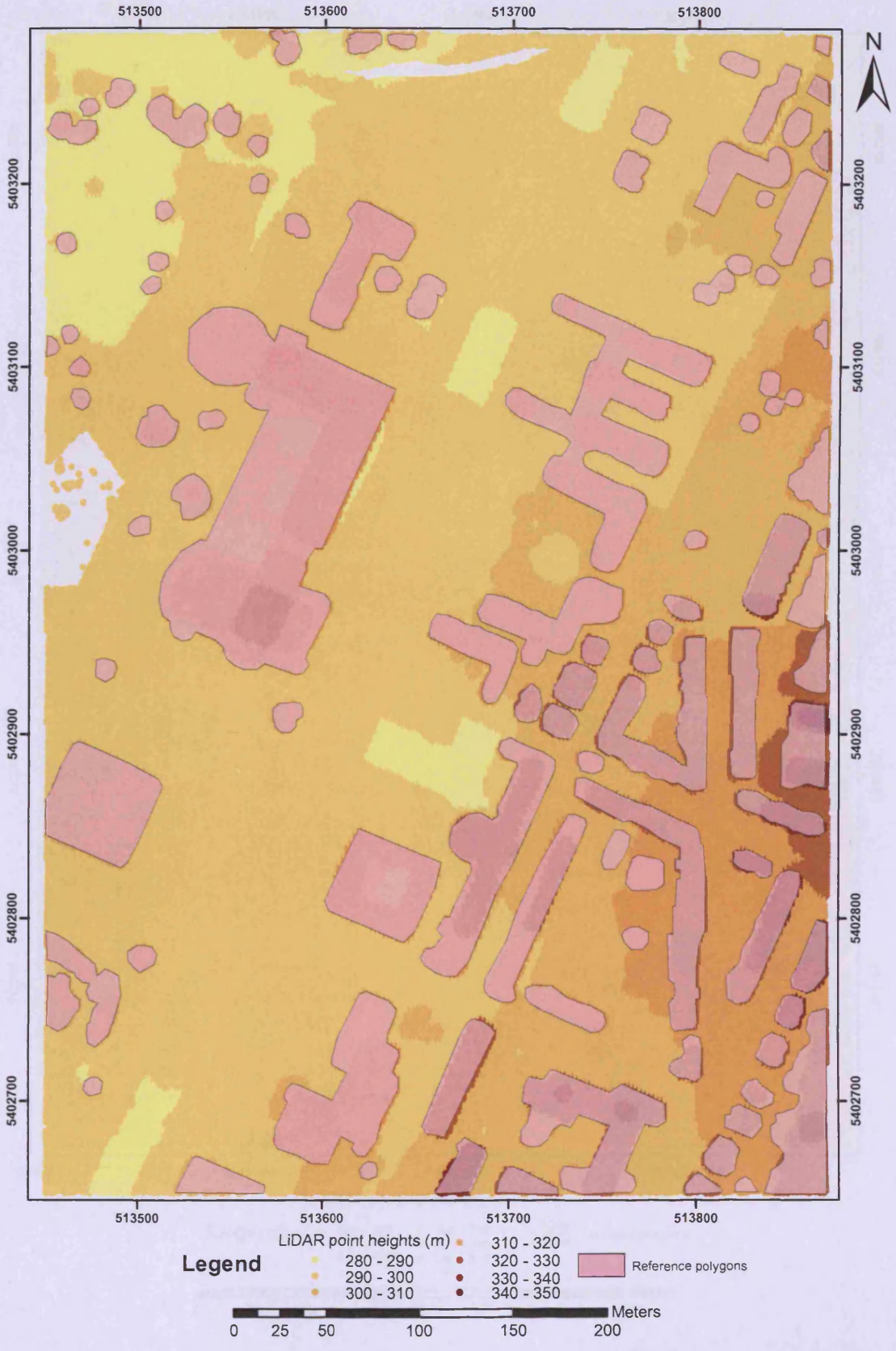


Figure 8.44 - The reference scene model for Site1 of Stuttgart, obtained by digitization of the original LiDAR point cloud.

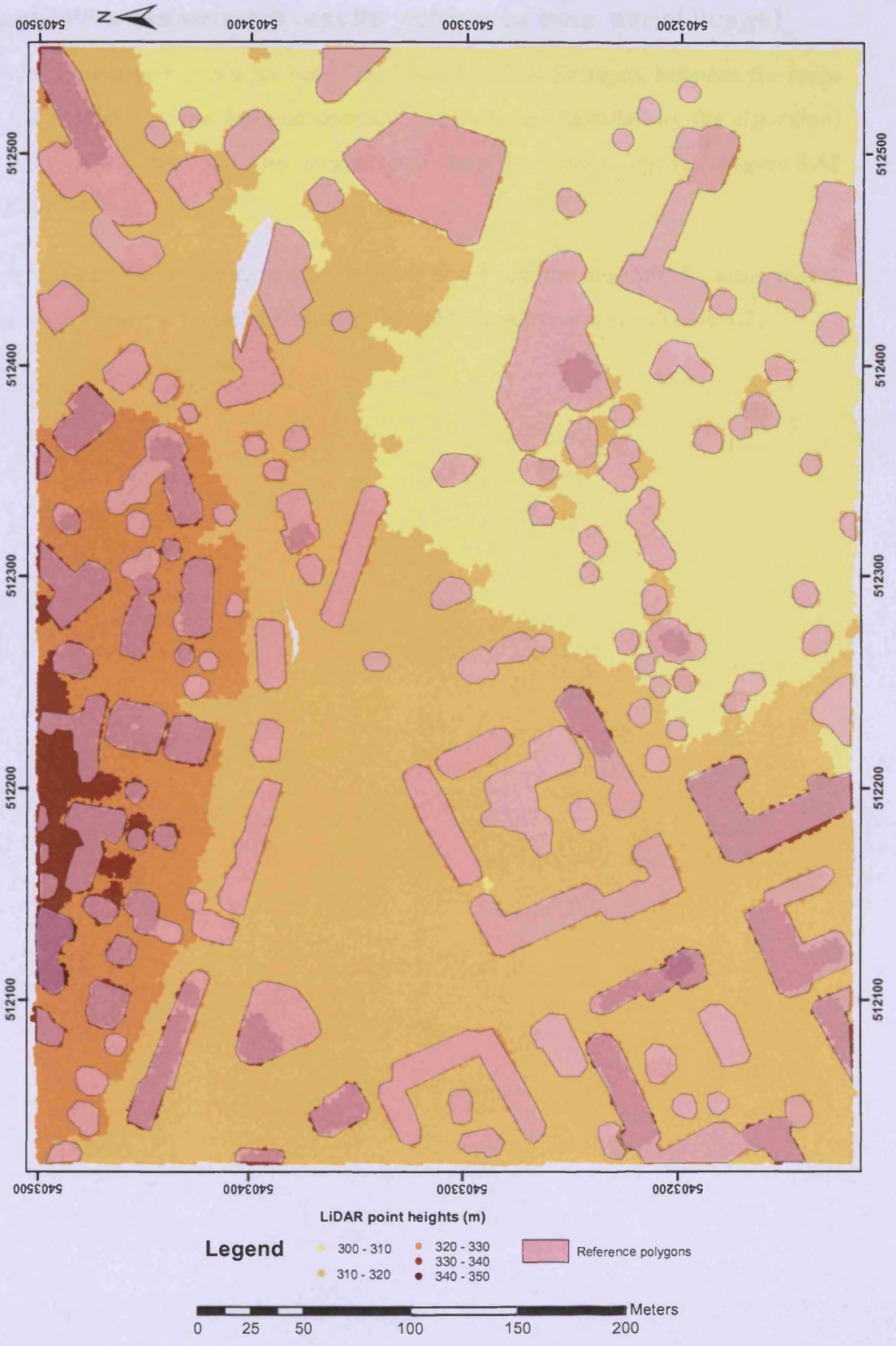


Figure 8.45 - The reference scene model for Site2 of Stuttgart, obtained by digitization of the original LiDAR point cloud.

8.3.3 Quantitative assessment results (reference data: aerial image)

An overlay was carried out for both Site 1 and Site 2 of Stuttgart, between the steep and flat polygons of the detected containment units (as classified by the algorithm) and the spatial feature polygons as generated from the aerial image (*vd.* Figure 8.42 and Figure 8.43).

The overlay of the reference scene model of Site 1 and the algorithm's scene model is shown in Figure 8.46, and the results obtained are summarised in Table 8.2.

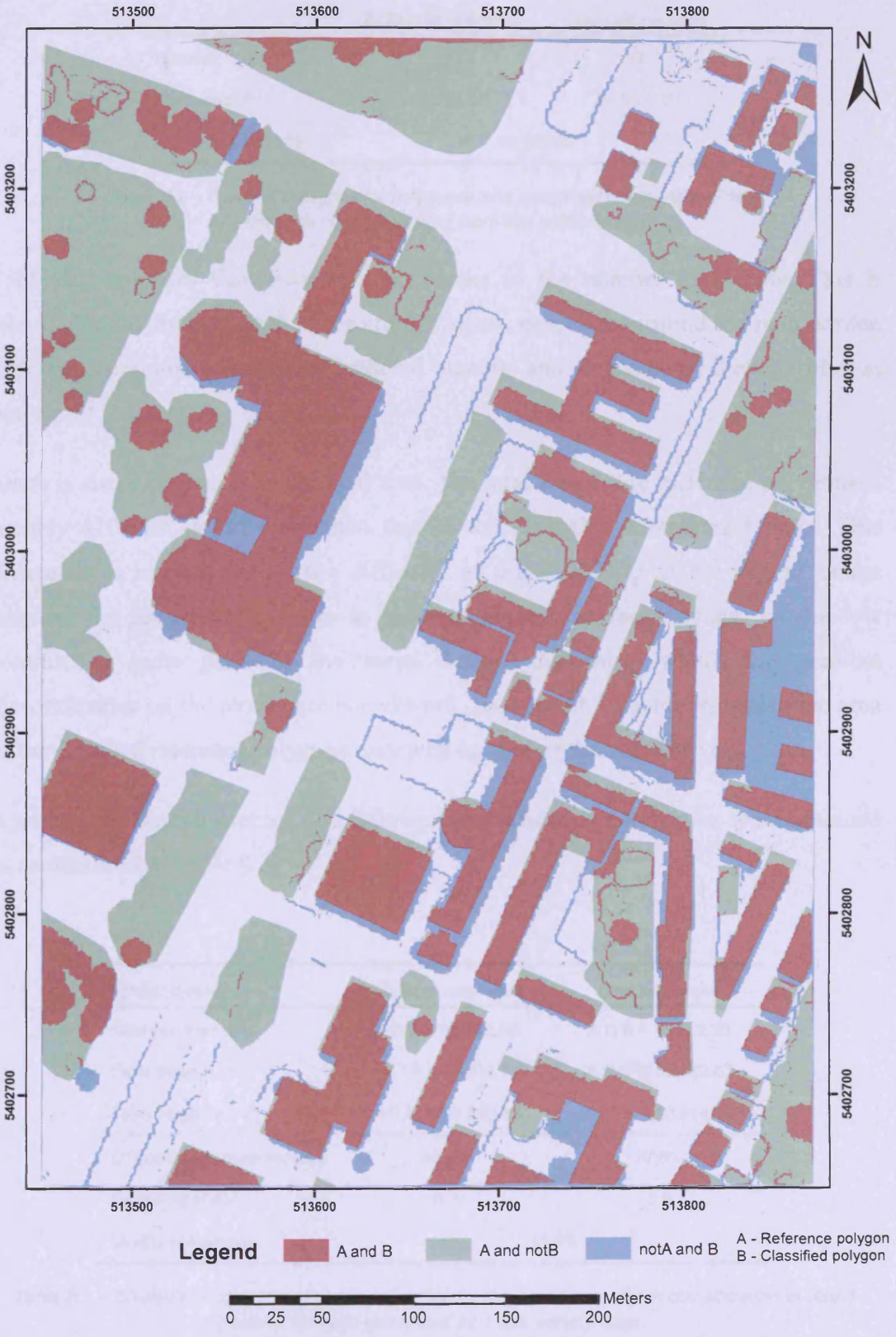


*Figure 8.46 – Assessment of the containment unit classification for Site 1 - reference data generated from the aerial image: (A **and** B) polygons represent areas where reference and classification coincide; (**not** A **and** B) are classified areas which are not part of the reference areas; (A **and not** B) are reference areas which are not classified.*

	Reference polygons	Containment units
Number	72	43
Total area (m^2)	125 552.97	84 537.02
Common area (m^2)	65 622.28	

Table 8.2 – General statistics for reference and classified polygons in Site 1. (Reference data generated from the aerial image)

Table 8.2 indicates that there is a difference in the number of objects. This is principally due to the fact that in some instances, especially around the map border, different containment units were gathered together and identified by the algorithm as one unit.

There is also a difference in the total area. The total area of the reference polygons is roughly 41000m^2 (49%) more than that of the classified containment units. This difference is mainly due to the difficulty in distinguishing in the digital image between the features off terrain as separate objects. In fact, because of the low contrast in some parts of the aerial image, discerning where the gradient discontinuities on the terrain are is awkward. As a result, in some instances the area of the digitized reference polygons may well be greater than actually is.

Applying the Shufelt metrics, the following classification parameters were obtained as summarised in Table 8.3.

Shufelt metrics	Producer accuracies	User accuracies
True positive (m^2)	A ∩ B = 65 622.28	A ∩ B = 65 622.28
False positive (m^2)	A' ∩ B = 18 914.74	A ∩ B' = 59 520.67
False negative (m^2)	A ∩ B' = 59 520.67	A' ∩ B = 18 914.74
Object detection percentage	52.4%	77.6%
Branching factor	0.3	0.9
Quality percentage	45.6%	

Table 8.3 – Shufelt's producer and user accuracies for the containment unit classification in Site 1 (Reference data generated from the aerial image)

An examination of both the producer and the user accuracies for the ODP and BF confirms the observations made above. The lower value registered for the producer ODP arises from the fact that a larger FN value is used in its computation. This

implies that a large area or number of polygons digitized from the aerial image did not exist as part of the classified containment units.

Furthermore, the producer BF is relatively low meaning that the classified data closely relate to the reference data; in fact, just a few polygons classified in the reference data as background correspond to objects in the algorithm's model. In contrast, the user BF is considerably high; this is explained by the large value of the user FP (*vd.* green areas in Figure 8.46), confirming that the reference polygons are larger in area than classified polygons.

The quality percentage is less than 50% in this case, and is greatly affected by the large values of FN and FP. All the discrepancies mentioned above seem to relate to some technical aspects in generating the reference data, such as image scale issues for digitizing purposes, the difficulties faced in the digitization process, and the fact that the image was not correctly ortho-rectified (a clear shift can be seen in Figure 8.27 and Figure 8.28 when overlaying the algorithm's scene model over the aerial photo).

The overlay of the reference scene model of Site 2 and the algorithm's scene model is shown in Figure 8.47, and the results obtained are summarised in Table 8.4.

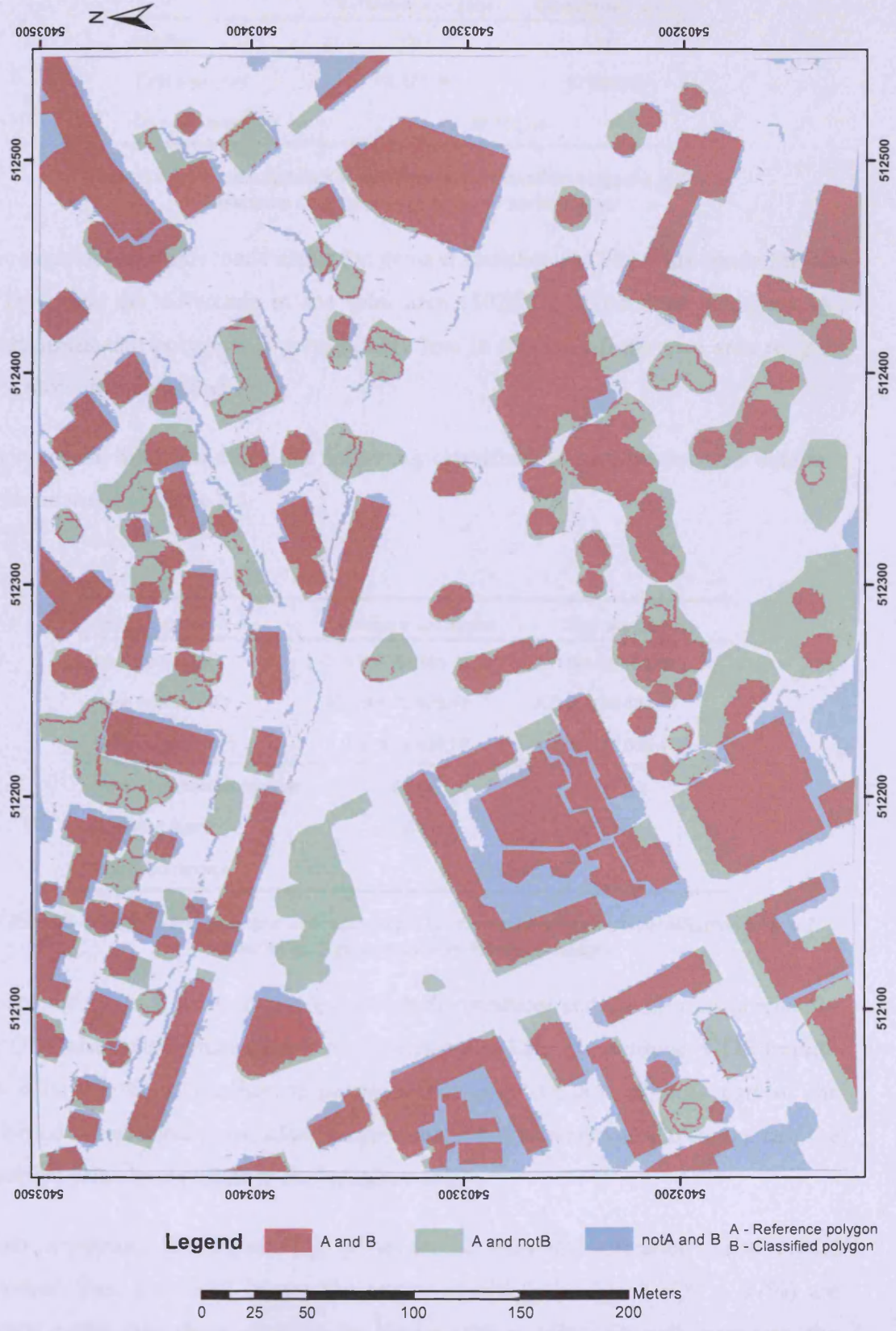


Figure 8.47 – Assessment of the containment unit classification for Site 2. (Reference data generated from the aerial image)

	Reference polygons	Containment units
Number	108	61
Total area (m^2)	75 727.10	67 698.93
Common area (m^2)	52 163.46	

Table 8.4 – General statistics for reference and classified polygons in Site 2. (Reference data generated from the aerial image)

The same observations made about the general statistics for Site 1 are made for Site 2. However, the difference in the total area (8028.17m^2) between reference and containment-unit polygons is considerably less in this case (reference area roughly 12% more than classified area).

Applying the Shufelt metrics, the following classification parameters were obtained as summarised in Table 8.5.

Shufelt metrics	Producer accuracies	User accuracies
True positive (m^2)	A ∩ B = 52 163.46	A ∩ B = 52 163.46
False positive (m^2)	A' ∩ B = 15 535.47	A ∩ B' = 23 534.70
False negative (m^2)	A ∩ B' = 23 534.70	A' ∩ B = 15 535.47
Object detection percentage	68.9%	77.1%
Branching factor	0.3	0.5
Quality percentage	57.2%	

Table 8.5 – Shufelt's producer and user accuracies for the containment unit classification in Site 2 (Reference data generated from the aerial image)

As noted for Site 1, an examination of both the producer and the user accuracies for the ODP also tells us that the lower value registered for the producer ODP implies that a large area or number of reference polygons did not exist as part of the classified containment units. Also, the producer BF is relatively low meaning that the classified data closely relate to the reference data.

When comparing the algorithm's scene model with the reference scene model generated from the aerial image, the results obtained for Site 2 (QP = 57%) are slightly better than those obtained for Site 1 (QP = 46%). Overall, however, the results obtained cannot be considered fully satisfactory (QP = 50% when combining results for Site 1 and Site 2). It is believed that the reason for this may not have been

the general performance of the algorithm, but the quality of the reference data generated from the aerial photo. The algorithm was designed to detect containment units. These however are not pictured in the aerial photo, and hence identifying entities that could be easily related to the detected containment units was not straightforward. As discussed in Chapter 6, it was assumed that where containment occurs within the UEB there is a high likelihood of a spatial feature being present. Thus, spatial features, like buildings and trees, discernable in the aerial photo were primarily used for the generation of reference polygons. But, the issues faced during the digitization of the aerial photo may have produced somewhat inaccurate reference data. In conclusion, a more realistic assessment can only be accomplished where higher quality truth data are used.

8.3.4 Quantitative assessment results (reference data: LiDAR data)

A second overlay was carried out for both Site 1 and Site 2 of Stuttgart, between the steep and flat polygons of the detected containment units (as classified by the algorithm) and the spatial feature polygons as generated from the original LiDAR point cloud (*vd.* Figure 8.44 and Figure 8.45).

At a glance, it can be seen in Figure 8.48 and Figure 8.49 that the polygons representing coincidence between the reference and the classified data are predominant, meaning that in both Site 1 and Site 2 most of the identified containment-unit areas coincide with spatial features.

The overlay of the reference scene model of Site 1 and the algorithm's scene model is shown in Figure 8.48, and the results obtained are summarised in Table 8.6.

Figure 8.48 – Assessment of the containment unit classification for Site 1. (Reference data generated from the original LiDAR point cloud)

	Reference polygons	Containment units
Number	101	43
Total area (m^2)	70 941.47	84 537.02
Common area (m^2)	66 293.88	

Table 8.6 – General statistics for reference and classified polygons in Site 1. (Reference data generated from the original LiDAR point cloud)

Table 8.6 indicates there are still a greater number of reference polygons than containment units. This difference is due to the reasons pointed out in section 8.3.3, and is actually greater here than in that case. Nevertheless, the total area of the algorithm's model is greater than the area of the reference model (19% more). The figures in Table 8.6 were used to compute Shufelt metrics. These were obtained as summarised in Table 8.7.

Shufelt metrics	Producer accuracies	User accuracies
True positive (m^2)	A ∩ B = 66 293.88	A ∩ B = 66 293.88
False positive (m^2)	A' ∩ B = 18 243.13	A ∩ B' = 4 567.87
False negative (m^2)	A ∩ B' = 4 567.87	A' ∩ B = 18 243.13
Object detection percentage	93.6%	78.4%
Branching factor	0.3	0.07
Quality percentage	74.4%	

Table 8.7 - Shufelt's producer and user accuracies for the containment unit classification in Site 1. (Reference data generated from the original LiDAR point cloud)

The lower value registered for the user ODP as compared to the producer ODP means that a large area of the algorithm's model did not exist as part of the reference model. This arises from the fact that a larger FN value is used in the computation of the user ODP. Most buildings were in close proximity to trees and this possibly led to larger containment-unit polygons classified by the algorithm, hence the large value for the user FN.

The producer BF is higher than the user BF; this is because the producer FP is higher than the user FP. As said above, the proximity of vegetation to buildings possibly resulted in an over sampling of containment-unit boundaries. In addition, some areas

situated to the east of Figure 8.48 (in blue) were missed from the reference data, and hence they are seen to be large contributors of a high producer FP.

Though it is expected that QP will be lower than the ODP, the producer ODP is higher than the QP by about 20%. The value of FP plays an important role in the difference between those two values: the higher the value of FP (therefore over sampling) the larger the difference between ODP and QP.

The overlay of the reference scene model of Site 2 and the algorithm's scene model is shown in Figure 8.49, and the results obtained are summarised in Table 8.8.

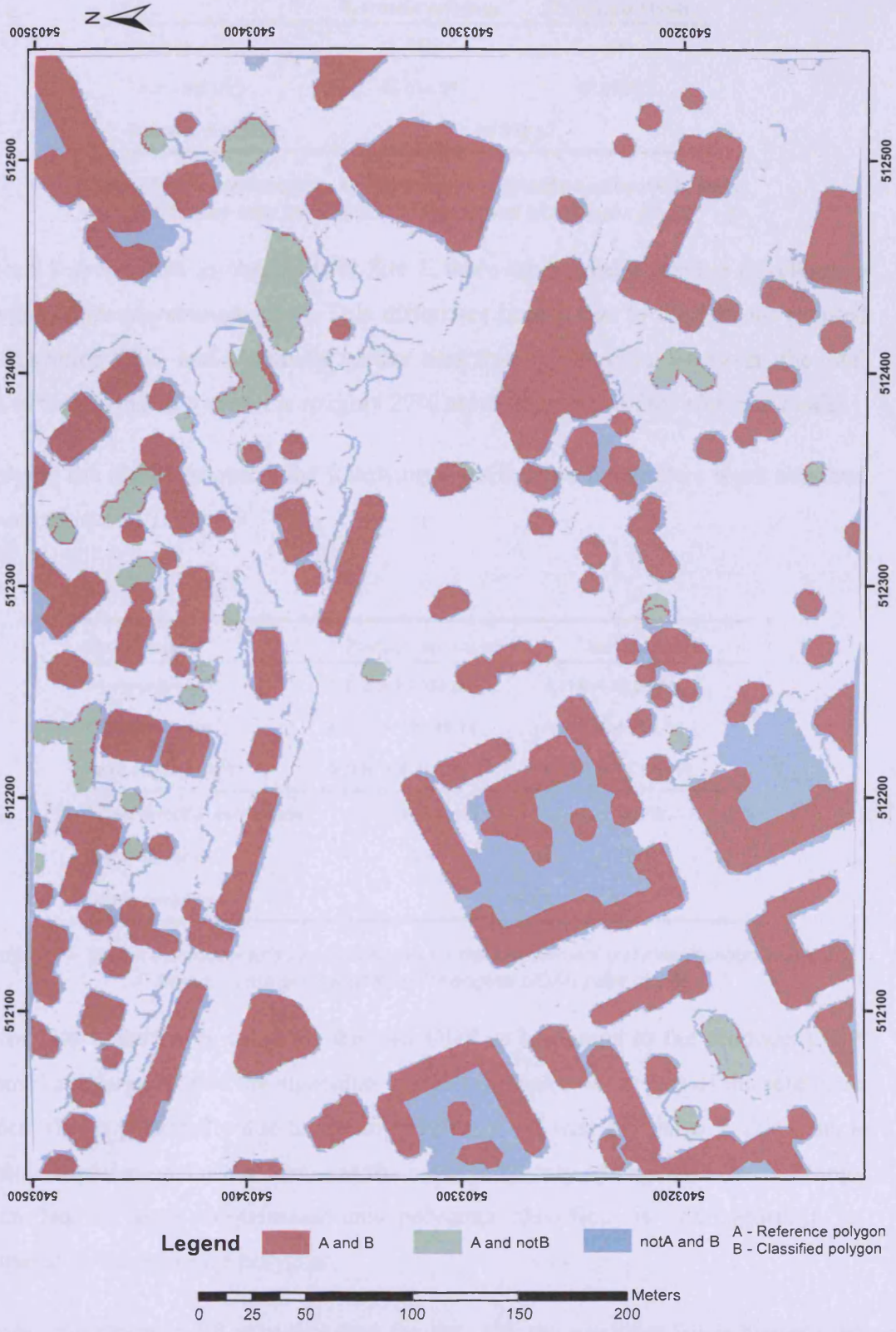


Figure 8.49 – Assessment of the containment unit classification for Site 2.
(Reference data generated from the original LiDAR point cloud)

	Reference polygons	Containment units
Number	125	61
Total area (m^2)	53 514.51	67 698.93
Common area (m^2)	48 552.82	

Table 8.8 – General statistics for reference and classified polygons in Site 2. (Reference data generated from the original LiDAR point cloud)

Table 8.8 shows that, as observed for Site 1, there are a greater number of reference polygons than containment units. This difference is also due to the reasons pointed out in section 8.3.3, and is actually greater here than in that case. However, the total area of the algorithm's model is roughly 27% more than that of the reference model.

Applying the Shufelt metrics, the following classification parameters were obtained as summarised in Table 8.9.

Shufelt metrics	Producer accuracies	User accuracies
True positive (m^2)	A ∩ B = 48 552.82	A ∩ B = 48 552.82
False positive (m^2)	A' ∩ B = 19 146.11	A ∩ B' = 4 916.55
False negative (m^2)	A ∩ B' = 4 916.55	A' ∩ B = 19 146.11
Object detection percentage	90.8%	71.7%
Branching factor	0.4	0.1
Quality percentage	66.9%	

Table 8.9 – Shufelt's producer and user accuracies for the containment unit classification in Site 2. (Reference data generated from the original LiDAR point cloud)

As for Site 1, the lower value for the user ODP as compared to the producer ODP means that a large area of the algorithm's model did not exist as part of the reference model. This is principally due to the large value of the user FN. Also in this case, a possible explanation for that fact was the close proximity of vegetation to buildings which led to larger containment-unit polygons classified by the algorithm as compared to the reference polygons.

Because the producer FP is higher than the user FP, the producer BF is higher than the user BF. As explained for Site 1, the proximity of vegetation to buildings possibly resulted in an over sampling of containment-unit boundaries. Moreover,

some areas situated especially to the south and southwest of Figure 8.49 (in blue) were missed from the reference data, and hence contributed to a high producer FP.

The producer ODP is higher than the QP by about 24%. As pointed out for Site 1, the high value of the producer FP is the principal reason for that difference (therefore over sampling and some areas missed out from the reference data).

When comparing the algorithm's scene model with the reference scene model generated from the original LiDAR point clouds, the results obtained for Site 1 (QP = 74%) are slightly better than those obtained for Site 2 (QP = 67%). The results for both Site 1 and Site 2 (QP = 71% when combining both sites) are better than those described in section 8.3.3, which was to some extent expected. Indeed, it is believed that the generation of reference polygons from the LiDAR point clouds produced more accurate reference data as compared to the generation process of reference polygons from the aerial photo. In spite of not providing any spatial feature boundaries, it was somewhat easier to detect in the LiDAR point clouds, colour coded in elevation range, the discontinuities on the terrain in terms of gradients.

8.4 Summary and discussion

In order to test its performance and versatility, the algorithm was applied to different kinds of unstructured data: LiDAR as well as photogrammetric data. The experiments accomplished have been presented and discussed in this chapter.

When applying the algorithm to the London (Kew) LiDAR dataset, the results obtained were not so meaningful in terms of urban topology analysis as compared to those obtained for the case study area. The complex pattern of this particular urban scene associated to the low resolution of the data, seem to explain that fact. Further experimentation with higher resolution LiDAR data of the same area was sought; however, such data were not available.

The algorithm was applied to higher resolution LiDAR data of two sites in the urban area of Stuttgart. Different experiments were accomplished with both raw and morphologically filtered data, and by considering different gradient thresholds in the preparation of the data process. A visual inspection of the results obtained suggested

that the algorithm performed best with morphologically filtered data when considering a 60° gradient thresholding: most of the containment units identified by the algorithm relate to individual urban features like buildings and trees.

A quantitative evaluation of the results above was carried out based upon a comparison of coincidences in areas. The assessment of the containment-unit classification relied upon a single aerial image as well as the original LiDAR point cloud (covering both test sites of Stuttgart), from which the reference polygons were digitized by on-screen digitization in ArcMap. Shadows in the aerial image, low image contrast, and poor image resolution led to inaccuracies in generating the reference polygons from the aerial photo (*vd.* Table 8.3 and Table 8.5). In contrast, in spite of the non-existence of spatial feature outlines in the LiDAR point cloud, the digitization of their boundaries was more accurate than in the aerial photo by following the gradient discontinuities on the terrain, and linking LiDAR points of the same height class (*vd.* Table 8.7 and Table 8.9). The combined Shufelt's metrics for both Site 1 and Site 2, for each kind of reference data, are summarised below in Table 8.10.

		Shufelt's metrics		
		Object detection percentage	Branching factor	Quality percentage
Site 1 & Site 2 (Reference data: aerial photo)	Producer accuracies	58.6%	0.3	50%
	User accuracies	77.4%	0.7	
Site 1 & Site 2 (Reference data: LiDAR data)	Producer accuracies	92.4%	0.3	71%
	User accuracies	75.4%	0.08	

Table 8.10 – Combined Shufelt's metrics for containment-unit classification for both Site 1 and Site 2.

The results of the quality assessment of the containment-unit classification confirmed the prediction that the digitization of the reference polygons from the aerial photo produced more inaccuracies. The overall quality percentage was 50% when the algorithm's scene model was compared to the reference model generated from the aerial photo. The facts that in that case the producer ODP is less than the user ODP, and also that the user BF is relatively high, confirm that a large area or number of polygons digitized from the aerial image did not exist as part of the classified containment units (*vd.* green polygons in both Figure 8.46 and Figure 8.47).

When comparing the algorithm's scene model with the reference scene model generated from the original LiDAR point clouds, the results obtained were better than those described above; Table 8.10 indicates that overall the quality percentage was 71%. This was in fact predicted when considering this reference scene model, as the generation of its polygons was more accurate than that from the aerial photo. A pertinent question that can be asked is how legitimate the generation of reference data from the source data is. This issue was actually the reason for taking different reference data, namely from the aerial photo. However, a more realistic assessment can only be accomplished where higher quality truth data are used.

In order to test the versatility of the algorithm, further experiments were carried out with other sources of unstructured data. Three photogrammetric datasets of three different surveyed areas were mainly used: Barton-Bendish (UK), Heerbrugg (Switzerland), and Santa Lucija (Malta). The somewhat unsatisfactory results obtained represent in this case a failure of the test data, not technically of the algorithm. In fact, in some instances the low resolution of the photogrammetric data was an issue; in others, the under sampling of buildings and of gradient discontinuities on the terrain also constituted an issue. Efforts should be made to use more suitable photogrammetric data for the purposes of this algorithm.

The experiments carried out with real world data and the results obtained, lead us to believe that overall the method for the analysis of urban topology worked well. In fact, the algorithm accomplished what it was designed for: the detection of units of containment. The success of the analysis process when dealing with real world data, depends on the combination of various factors: the spread of locations where the 3D points lie, the point density, and the gradient threshold used in the binary classification of the TIN facets.

After describing the experiments carried out with real world data and discussing the results obtained, next chapter concludes this thesis by taking a broad look at the research conducted and its findings. An outlook on recommendations for future research and practical applications is also given.

9 Conclusions

This chapter concludes the research work described in this thesis. A broad look at the research conducted and its findings is taken in section 9.1: a reference to the main purpose of the study is made; a discussion on whether the initial aims were matched is carried out; the study outcome and its limitations are pointed out; the value of this study is also discussed and future practical applications are identified. Finally, some recommendations for further improvements are drawn in section 9.2; an outlook on the next steps to extend the scope of this study is also given.

9.1 General discussion

9.1.1 Purpose of the study

The study described in this thesis has taken a step in the direction of the use of spatial topology for the analysis and understanding of complex urban scenes.

A graph-theoretic approach was proposed to analyse entities within the urban environment. More precisely, graph theory was applied in the interpretation of the urban spatial topology based on the extension of the standard notion of the spatial relation of adjacency to that of containment: the containment-first search algorithm. Although geometric information is to some extent implicit in the designed methodology, this study sought to take a purely topological approach and investigate how far it would be possible to go just by looking at topological relationships between spatial objects.

In addition, this research attempted to work with initially unstructured geospatial data in order to study whether it would be possible to produce higher-level information. No prior knowledge of the spatial entities was assumed. The graph-based technique developed was applied to different LiDAR as well as photogrammetric geospatial datasets of urban areas.

9.1.2 Aims and objectives

In order to achieve the purposes mentioned in the previous section, several tasks were taken into consideration and accomplished:

- The initial problem was viewed as a general task of finding higher-level structures in the initial arbitrary collection of lower level details.
- Structured information translated into containment units was retrieved from the original data. This was achieved by:
    - identifying more meaningful structures within the initial random collection of objects;
    - and by understanding their spatial arrangement in terms of the topological relationships of adjacency, containment and touching between polygons.
- The investigation of the topological relationships between spatial objects was carried out in the context of the whole spatial scene rather than within their individual neighbourhood.
- For the purposes stated above, a higher-order topological analysis method was designed. The use of graph-theoretic approaches in the design of such a method was explored; in particular, the merits of depth-first and breath-first graph searching methods were investigated - the respective algorithm was implemented in C.
- This study sought to propose a method as much as possible independent of the test environment. However, it must be acknowledged that the unstructured data initially used (especially LiDAR data) influenced to some extent the design and implementation of the algorithm.
- The methods investigated were extended towards the visualisation of the resulting graphs of adjacencies, and representation of the urban scene topology. In this way, the higher-order topological analysis method was linked back to the original environment (*i. e.* the map of spatial objects). For this purpose, an interactive tool was developed in ArcMap – this was implemented in VBA using ESRI's ArcObjects.

9.1.3 Research outcome

Given the general complexity of an urban environment, analysing and understanding the urban scene are particularly laborious and time consuming tasks. The objective of working with unstructured data made those tasks more awkward and constituted a challenging research question. The study reported in this thesis proved that when working in these circumstances is still possible to retrieve higher-level information.

Some of the experiments (especially those with morphologically filtered data of Stuttgart) confirmed that the containment units detected by the algorithm are strongly related to the existence of spatial features, such as buildings and trees.

To go further and derive more detailed information from initially unstructured geospatial data, bringing in other sorts of information, such as related to geometry, may be necessary. Even so, this study fundamentally confirmed that topology is a central defining feature in a GIS, and showed that it is possible to retrieve higher-level information just by looking at topological relationships between spatial objects.

The present research also offers clear evidence of the value of graph theory in revealing the spatial arrangement of geographical entities. This is in substantial agreement with the authors mentioned throughout Chapters 2 and 3 (including Laurini and Thompson, 1992). The merits of both DFS and BFS graph searching algorithms in analysing the structure of the urban spatial topology were examined. The investigation revealed in particular how the BFS spanning tree is more meaningful in terms of the urban scene, and how the tree's branches potentially relate to the existence of urban features.

9.1.4 Limitations

It is readily acknowledged that this research is exploratory and there are issues namely with regards to the test environment used and to the methodology followed in the data preparation process.

As an example scenario, LiDAR data were used to develop and test the graph-based technique conceived in this work. Although the development of a method as much as possible independent of the test environment was sought, it is believed that the

LiDAR environment inevitably influenced the design of the graph-theoretic approach proposed.

In the data preparation process, the characteristics of LiDAR data led to the consideration of a binary classification of the TIN facets, based in particular upon their gradients. This was principally because it was assumed that the steep polygons would outline the spatial features, like buildings and trees. In these circumstances, the steep polygons may be seen as the interface ones separating flat polygons, which in turn may be seen as the main polygons. However, this classification is arguable as it does not correspond to standard GIS vector-based data, where normally lines separate polygons. As a consequence, the characteristics of the binary classification used make awkward the extension of the method proposed towards its application to structured data. In a certain way, these aspects restrict the extent to which the findings can be generalised.

9.1.5 Value and possible applications

The work reported in this thesis builds on work conducted in the fields of topology and graph theory. It combines and enhances some of their aspects in a new cocktail which should lead to a better understanding of complex urban scenes, especially when the starting point is unstructured data.

Though the algorithm was not designed for urban feature extraction, some of the results obtained with LiDAR data (especially morphologically filtered data of Stuttgart) reveal that this technique has promised as a tool that could be extended and applied in automatic classification of raw LiDAR data as buildings, ground and vegetation (Kim and Muller, 1999; Nardinocchi *et al.*, 2003; Forlani *et al.*, 2006).

Furthermore, in spite of the limitations discussed in the previous section, the results suggest that the concepts drawn in this thesis as well as the algorithm implemented may well serve as the basis for other applications for automatic analysis of spatial datasets, such as: analysis of image data (Nichol, 1990); analysis of settlement structures (Anders *et al.*, 1999); automation of the land use mapping process for urban areas (Bauer and Steinnocher, 2001; Barr and Barnsley, 2004).

9.2 Recommendations and future research

As noted in section 9.1.4, the test environment had a certain influence in the data preparation process. The classification of the TIN facets was based on a single attribute: their gradient. The use of other kinds of data will possibly lead to the consideration of another attribute, or a combination of attributes. Also worth of investigation is whether a binary classification has the same meaning when applied to other sorts of spatial data (not necessarily unstructured).

Moreover, the characteristics of the test data suggested a binary classification; could more than two classes have been considered? Clearly, the problem is simplified by taking into account only two gradient classes. Hypothetically, more classes could have been considered but the algorithm would not work in those circumstances; indeed, the method developed applies only to bipartite graphs of adjacencies.

Map edge effects mentioned throughout Chapter 8 take place because steep-polygon rings include sliver polygons lying around the map edges. The sliver polygons end up linking and gathering in a single unit different containment units. The performance of the algorithm should improve if the edge effects are tackled by removing the sliver polygons in the preparation of data process. An unsuccessful attempt was actually made to address the issue. Indeed, the problem is technically complex and further study is needed.

In a further step, the code of the visualisation tool for the spatial representation of the urban topology should be modularized to become independent from the interface itself. Once this task is done, the functionalities of graph visualisation and exploration can be incorporated in any GIS application.

Further still, with regards to the visualisation tool, challenging questions like how one visualises spatial relationships of objects, and how such tools should be developed to aid the interpretation of complex spatial scenes (such as urban areas), remain open. Further investigation is required to answer them.

Because the photogrammetric data available were found not suitable enough for the purposes of the algorithm implemented, it would be beneficial to replicate the approach outlined in this study on more photogrammetric datasets. Further

experimentation on other sources of unstructured data would also be interesting in order to test the versatility of the algorithm.

Future work should entail the investigation and implementation of other rules for the graph analysis process, eventually leading to a possible aggregation of graph vertices into identified meaningful structures. Albeit the double depth-first graph search is a classic algorithm used to identify strongly connected components in directed graphs (Cormen *et al.*, 1990), a possible extension of this algorithm for the purposes above seems worth of investigation. The identified meaningful structures, in turn, should be clustered into homogenous regions (Forberg and Raheja, 2002). After the delineation of cluster shapes, an analysis process may have to be accomplished, either by pattern recognition or interpretation procedures (Toussaint, 1980b). The aim of the ultimate cluster shapes analysis would be the retrieval of higher-level information, *e.g.* sets of buildings, vegetation areas, and say land-use parcels.

References

ACKERMANN, F., 1999. Airborne laser scanning – present status and future expectations. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 64-67. Elsevier Science Ltd.

ALÇADA-ALMEIDA, L.; TRALHÃO, L.; SANTOS, L.; COUTINHO-RODRIGUES, J. [in press, to be published in 2007]. A multiobjective approach for locating shelters and developing evacuation routes for emergencies in urban areas. In *Geographical Analysis*, Special Issue in Memory of C. ReVelle. Blackwell Publishing.

DE ALMEIDA J.-P.; MORLEY J. G.; DOWMAN I. J. [in press, to be published in 2007]. Graph theory in higher order topological analysis of urban scenes. In *Computers, Environment and Urban Systems*. Elsevier Science Ltd.

ANDERS, K.-H.; SESTER, M.; FRITSCH, D., 1999. Analysis of settlement structures by graph-based clustering. *Semantische Modellierung, SMATI 99*, 41-49. Munich (Germany).

ARONOFF, S., 1989. *Geographic Information Systems: A Management Perspective*. WDL Publications, Ottawa (Ontario, Canada).

AXELSSON, P., 1999. Processing of laser scanner data – algorithms and applications. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 138-147. Elsevier Science Ltd.

AYUGI, S., 2006. *Extraction of Geographic Features from LiDAR data*. PhD thesis (unpublished). Department of Geomatic Engineering, University College London (England, UK).

BAFNA, S., 2003. Space syntax – A brief introduction to its logic and analytical techniques. In *Environment and behavior*, Vol. 35, No. 1: 17-29. Sage Publications.

BALLARD, D.H.; BROWN, C.M., 1982. *Computer Vision*. Prentice Hall, Englewood Cliffs.

BALTSAVIAS, E.P., 1999a. A comparison between photogrammetry and laser scanning. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 83-94. Elsevier Science Ltd.

BALTSAVIAS, E.P., 1999b. Airborne Laser Scanning: existing systems and firms and other resources. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 164-198. Elsevier Science Ltd.

BALTSAVIAS, E.P., 1999c. Airborne Laser Scanning: basic relations and formulas. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 199-214. Elsevier Science Ltd.

BARNSLEY, M.J.; BARR, S.L., 1998. Distinguishing urban land-use categories in fine spatial resolution land-cover data using a graph-based, structural pattern recognition system. *Computers, Environment and Urban Systems*, Vol. 21, No. 3/4: 209-225. Elsevier Science Ltd.

BARNSLEY, M.J., 2003. Mapping Land-Use in Urban Areas [online]. Available from http://stress.swan.ac.uk/~mbarnsle/research/urban.htm [Accessed 24 January 2003].

BARR, S.L.; BARNSLEY, M.J., 1996. Inferring urban land-use from satellite sensor images using kernel-based spatial reclassification. *Photogrammetric Engineering and Remote Sensing*, No. 62: 949-958.

BARR, S.L.; BARNSLEY, M.J., 1997. A region-based, graph-theoretic data model for the inference of second-order thematic information from remotely-sensed images. *International Journal of Geographical Information Science*, Vol. 11, No. 6: 555-576. Taylor & Francis Ltd.

BARR, S.L.; BARNSLEY, M.J., 2004. Characterising the structural form of an urban system using built-form connectivity model concepts. In *Spatial Modelling of the Terrestrial Environment* (KELLY, R.E.J.; DRAKE, N.A.; BARR, S.L., eds.): 201-226. John Wiley, Chichester (England, UK).

BAUER, T.; STEINNOCHER K., 2001. Per-parcel land use classification in urban areas applying a rule-based technique. *GeoBIT/GIS 6*: 24-27.

BENEDIKT, M., 1979. To take hold of space: isovists and isovist fields. In *Environment and behavior*, B 6: 47-65. Sage Publications.

BLAIR, J.B.; RABINE, D.L.; HOFTON, M.A., 1999. The laser vegetation imaging sensor: a medium-altitude, digitisation-only, airborne laser altimeter for mapping vegetation and topography. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 115-122. Elsevier Science Ltd.

BOLLOBÁS, B., 1998. *Modern Graph Theory*. Springer-Verlag, New York (USA).

BOLLOBÁS, B., 2001. *Random Graphs* (2nd ed.). Cambridge University Press, Cambridge (England, UK).

BONHAM-CARTER, G.F., 1996. *Geographic Information Systems for Geoscientists: Modelling with GIS*. Pergamon Press, Ottawa (Ontario, Canada).

BORGATTI, S.P.; EVERETT, M.G.; FREEMAN, L.C., 2002. *Ucinet for Windows: Software for Social Network Analysis*. Analytic Technologies, Harvard (MA, USA).

BUNN, A.G.; URBAN, D.L.; KEITT, T.H., 2000. Landscape connectivity: A conservation application of graph theory. *Journal of Environmental Management*, no.59, 265-275. Academic Press (USA).

BURROUGH, P.A.; MCDONNELL, R. A., 1998. *Principles of Geographical Information Systems*. Oxford University Press, Oxford (UK).

CARLSON, E., 1987. Three dimensional conceptual modeling of subsurface structures. *Technical Papers of ACSM-ASPRS Annual Convention*, 188-200. Baltimore (USA).

CHANG, K.-T., 2005. *Programming ArcObjects with VBA: A Task-Oriented Approach*. CRC Press LLC, Boca Raton (Florida, USA).

CHARTRAND, G.; LESNIAK, L., 1986. *Graphs & Digraphs* (2nd ed.). Wadsworth & Brooks/Cole.

CHRISMAN, N., 1997. *Exploring Geographic Information Systems*. John Wiley & Sons, New York (USA).

CLARKE, K. C., 1990. *Analytical and Computer Cartography*. Prentice Hall, Englewood Cliffs (USA).

CLEMENTINI, E.; DI FELICE, P.; OOSTEROM, P., 1994. A small set of formal topological relationships suitable for end-user interaction. Lecture notes in Computer Science [online]. In *Proceedings of the III Symposium on Advances in Spatial Databases* (ABEL, D. J.; OOI B. C., eds.): 277-295. Springer Verlag, Singapore. Available from http://skyblue.csd.auth.gr/~alex/sdb/artSSD93.pdf [Accessed 22 February 2006].

COHN, A.; BENNETT, B.; GOODAY, J.; GOTTS, N., 1997. Qualitative spatial representation and reasoning with the Region Connection Calculus (RCC8). In *Geoinformatica*, Vol.1: 1-44. Kluwer Academic Publishers, Boston (MA, USA).

COHON, J. L., 1978. *Multiobjective Programming and Planning*. Academic Press, Orlando (Fla-USA).

COOKE, D. F.; MAXFIELD, W. H., 1967. The development of a geographic base file and its uses in mapping. In *Urban and Regional Information Systems Association*, Vol.5: 207-218.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L., 1997. *Introduction to Algorithms* (18th printing), 23. The Massachusetts Institute of Technology, Boston (MA-USA).

COUTINHO-RODRIGUES, J.; CLÍMACO, J.; CURRENT, J., 1994. A PC-based interactive decision-support system for a two objective direct delivery problems. *Journal of Business Logistics*, Vol.15, No.1: 305-322. Council of Logistics Management, Oak Brook (Illinois, USA).

COUTINHO-RODRIGUES, J.; CURRENT, J.; CLÍMACO, J.; RATICK, S., 1997. Interactive spatial decision-support system for multiobjective hazardous materials location-routing problems. *Transportation Research Record*, No.1602: 101-109. National Academy Press, Washington (D.C., USA).

COWEN, D. J., 1988. GIS versus CAD versus DBMS: what are the differences? *Photogrammetric Engineering & Remote Sensing*, 54, 1551-1555.

CURRENT, J.; RATICK, S., 1995. A model to assess risk, equity, and efficiency in facility location and transportation of hazardous materials. *Location Science*, Vol. 3: 187-203.

DAVENHALL, A. C., 1997. *An Introduction to Visualisation Software for Astronomy - Starlink Guide 8* [online]. Available from http://www.starlink.rl.ac.uk/star/docs/sg8.htx/node9.html [Accessed 10 May 2005].

DEMERS, M. N., 1997. *Fundamentals of Geographic Information Systems*. John Wiley & Sons, New York (USA).

DOWNS, R.; STEA, D., 1977. *Maps in Minds: reflections on cognitive mapping*, pp 284. Harper & Row Publishers, New York (USA), London (England, UK).

DYKES, J.; MACEACHREN, A. M.; KRAAK, M.-J., 2005. *Exploring geovisualization*. In *Exploring Geovisualization* (DYKES, J.; MACEACHREN, A. M.; KRAAK, M.-J., eds), 1: 3-19. Elsevier Science Ltd.

EDELSBRUNNER, H.; KIRKPATRICK, D.G.; SEIDEL, R., 1983. On the shape of a set of points in the plane. In *IEEE Transactions on Information Theory*, Vol.29, no.4: 551-559.

EGENHOFER, M. J., 1991. Reasoning about binary topological relations. Proceedings of the *Second Symposium on Large Spatial Databases, SSD'91* (GUNTHER, O.; SCHEK H. J., eds). Zurich (Switzerland). Lecture notes in Computer Science 525: 143-160.

EGENHOFER, M. J., 1994. Topological similarity. Proceedings of *FISI Workshop on the Topological Foundations of Cognitive Science*, Vol.37 of *Reports of the Doctoral Series in Cognitive Science*. University of Hamburg (Germany).

EGENHOFER, M. J.; HERRING, J. R., 1990. Categorizing binary topological relations between regions, lines, points in geographic databases [online]. *Technical Report, National Center for Geographic Information and Department of Survey Engineering, Department of Computer Science, University of Maine (USA).* Available from http://www.spatial.maine.edu/~max/9intReport.pdf [Accessed 15 July 2005].

EGENHOFER, M. J.; FRANZOSA, R. D., 1991. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, Vol.5, No.2: 161-174. Taylor & Francis Ltd.

EGENHOFER, M. J.; FRANZOSA, R. D., 1995. On the equivalence of topological relations. *International Journal of Geographical Information Systems*, Vol.9, No.2: 133-152. Taylor & Francis Ltd.

EGENHOFER, M. J.; CLEMENTINI, E.; DI FELICE, P., 1994. Evaluating inconsistencies among multiple representations. *Sixth International Symposium on Spatial Data Handling*: 901-920. Edinburgh (Scotland, UK).

ELLUL, C., 2004. *Extending 2D SQL to topological queries in 3D*. MPhill/PhD Transfer Report (unpublished). Department of Geomatic Engineering, University College London (England, UK).

ENVIRONMENT AGENCY (EA), 1997. Evaluation of the LiDAR technique to produce elevation data for use within the agency. *R&D Summary*. National Centre for Environmental Data and Surveillance, Environmental Agency.

ENVIRONMENT AGENCY (EA), 1998. Airborne light detection and ranging feasibility study. *R&D Technical SummaryES41*. National Centre for Environmental Data and Surveillance, Environmental Agency.

ESRI, 1995. *Understanding GIS, The ARC/INFO Method*, Lesson2. ESRI, Inc., Cambridge (England, UK).

ESRI, 2004. *ArcGIS Topology* [online]. Available from http://www.esriuk.com/products/support/ArcGIS%20UK%20EXT/Topology.asp [Accessed 10 June 2005].

ESRI, 2005. *GIS Topology – An ESRI White Paper* [online]. Available from http://www.esri.com/library/whitepapers/pdfs/gis_topology.pdf [Accessed 28 April 2006].

EYTON, J.R., 1993. Urban land-use classification and modelling using cover-type frequencies. *Applied Geography*, Vol. 13: 111-121.

EUROSENSE, 2003. LiDAR/Airborne laser scanning [online]. Available from http://www.eurosense.com/Frame3.htm [Accessed 16 July 2003].

FORBERG, A.; RAHEJA, J.L., 2002. Generalization of 3D settlement structures on scale-space and structures recognition. Work Report, Institut für Photogrammetrie und Fenerkundung (Universität der Bundeswehr München), Lehrstuhl für Kartographie (Technische Universität München), Munich (Germany).

FORLANI, G.; NARDINOCCHI, C.; SCAIONI, M.; ZINGARETTI, P., 2006. Complete classification of raw LiDAR data and 3D reconstruction of buildings. In *Pattern Analysis and Applications*, Vol.8, 4: 357-374. Springer-Verlag, London (England, UK).

FRANK, A.; KUHN, W., 1986. Cell graphs: a provable correct method for the storage of geometry. In *Proceedings of Second International Symposium on Spatial Data Handling* (MARBLE, D., ed). Seattle (Washington, USA).

GIBBONS, A., 1989. *Algorithmic Graph Theory*. Cambridge University Press, Cambridge (England, UK).

GROSS, J.; YELLEN, J., 1999. *Graph Theory and Its Applications*. CRC Press LLC, Boca Raton (Florida, USA).

HAARSLEV, V.; MÖLLER, R., 1997. A qualitative reasoner: progress report. Proceedings of the *11th Symposium on Qualitative Reasoning* (IRONI L., ed). Tuscany (Italy).

HANSEN, P., 1980. Bicriterion path problems. In *Lecture Notes in Economics and Mathematical Systems* (BECKMAN, M.; KUNZI, H., Eds.), Vol. 177: 109-137. Springer, Berlin (Germany).

HEIJDEN, F. VAN DER, 1994. *Image Based Measurement Systems: Object Recognition and Parameter Estimation*. John Wiley, Chichester (England, UK).

HELMAN, J.; HESSELINK, L., 1991. Visualizing Vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3): 36-46.

HILLIER, B.; PENN, A.; HANSON, J.; GRAJEWSKI, T.; XU, J., 1993. Natural movement: configuration and attraction in urban pedestrian movement. In *Environment and Planning*, B 20: 29-66.

HUG, CH., 1996. Entwicklung und Erprobung eines abbildenden Laseraltimeters für den Flugeinsatz unter Verwendung des Mehrfrequenz-Phasenvergleichsverfahrens. PhD dissertation, Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften, Reihe C, Heft Nr. 457.

INTERGRAPH, 2004. GeoMedia Professional [online]. Available from http://imgs.intergraph.com/gmpro/ [Accessed 10 June 2005].

IRISH, J.L.; LILLYCROP, W.J, 1999. Scanning laser mapping of the coastal zone: the SHOALS system. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 123-129. Elsevier Science Ltd.

JIANG, B.; CLARAMUNT, C.; LINK, L.; KLARQVIST, B., 2000. An integration of space syntax into GIS modelling urban spaces. In *International Journal of Applied Earth Observation & Geoinformation*, Vol. 2, No. 3/4: 161-171.

JONES, M.; FANEK, M., 1997. Crime in the urban environment. In Proceedings, *First International Symposium on Space Syntax* (HILLIER B., Ed), 16-18 April: 25.1-25.11. University College London, (England, UK).

KASANEN, E.; OSTERMARK, R.; ZELENY, M., 1991. Gestalt system of holistic graphics: New management support view of MCDM. *Computers & Operations Research*, Vol. 18, No. 2: 233-239.

KELLEY, A.; POHL, I., 1990. *A Book on C - Programming in C* (2nd ed). Benjamin Cummings, Redwood City (CA, USA).

KERNIGHAN, B.; RITCHIE, D., 1988. *The ANSI C Programming Language* (2nd ed). Prentice Hall, Englewood (NJ, USA).

KIM, J.R.; MULLER, J.-P., 2002. 3D Reconstruction from very high resolution satellite stereo and its application to object identification. *Symposium on Geospatial Theory, Processing and Applications*. Ottawa (Canada).

KIM, J.R.; MULLER, J.-P., 2006. Automated tree and building detection in residential areas from stereo IKONOS images. *ISPRS Commission IV, WG IV/3*.

KIRKPATRICK, D.G.; RADKE, J.D., 1985. A framework for computational morphology. *Computational Geometry* (TOUSSAINT, G.T., Ed.), 217-248. North- Holland, Amsterdam (The Netherlands).

KLIMBERG, R., 1992. GRADS: A new graphical display system for visualizing multiple criteria solutions. *Computers & Operations Research*, Vol. 19: 707-711.

KOKKAS, N.; DOWMAN, M., 2006. Fusion of airborne optical and LiDAR data for automated building reconstruction. In *American Society for Photogrammetry and Remote Sensing*, May: 1-7. Reno (Nevada, USA).

KRABILL, W.; COLLINS, J.; LINK, L.; SWIFT, R.; BUTLER, M., 1994. Airborne laser topographic mapping results. *Photogrammetry Engineering and Remote Sensing*, 50: 685-694.

KRABILL, W.; THOMAS, R.; JEZEK, K.; KUIVINEN, K.; MANIZADE, S., 1995. Greenland ice sheet thickness changes measured by laser altimetry. *Geophysical Research Letters*, 22: 2341-2344.

KRÜGER, MÁRIO, 1979a. An approach to built-form connectivity at an urban scale: system description and its representation. In *Environment & Planning* B, Vol. 6, 67-88.

KRÜGER, MÁRIO, 1979b. An approach to built-form connectivity at an urban scale: variations of connectivity and adjacency measures amongst zones and other related topics. In *Environment & Planning* B, Vol. 6, 305-320.

KRÜGER, MÁRIO, 1980. An approach to built-form connectivity at an urban scale: relationships between built-form connectivity, adjacency measures, and urban spatial structure. In *Environment & Planning* B, Vol. 7, 163-194.

KRÜGER, MÁRIO, 1981a. An approach to built-form connectivity at an urban scale: modelling the distribution of partitions and built-form arrays. In *Environment & Planning* B, Vol. 8, 41-56.

KRÜGER, MÁRIO, 1981b. An approach to built-form connectivity at an urban scale: modelling the disaggregation of built forms types. In *Environment & Planning* B, Vol. 8, 57-72.

KRÜGER, MÁRIO, 1989. On node and axial maps: distance measures and related topics. Paper presented at *European Conference on the Representation and Management of Urban Change*, 28-29 September 1999. Cambridge (England, UK).

KRUSE, R.L.; LEUNG, B.P.; TONDO, C.L., 1991. *Data Structures and Program Design in C.* Prince Hall. New York (USA).

LAURINI, R.; THOMPSON, D., 1992. *Fundamentals of Spatial Information, 5.* Academic Press, London (England, UK).

LINDENBERGER, J., 1993. Laser-Profilmessung zur topographischen Geländeaufnhme. PhD dissertation, Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften, Reihe C, Heft Nr. 400.

LONGLEY, P.; GOODCHILD, M.; MAGUIRE, D.; RHIND, D., 2005. *Geographical Information Systems and Science (2nd Edition)*, Part IV, 13: 289-313. John Wiley & Sons Ltd, London (England, UK).

MARK, D.M., 1977. *Topological Randomness of Geographic Surfaces of Geomorphic Surfaces.* Technical Report #15, ONR Contract N00014-75-CO886.

MAGUIRE, D.; DANGERMOND, J., 1991. The functionality of GIS. In *Geographical Information Systems: Principles and Applications* (MAGUIRE, D.; GOODCHILD, M.; RHIND, D.; Eds.), 319-335. Longman Scientific & Technical, Harlow (USA).

MCDONNELL, R.; KEMP, K., 1995. *International GIS Dictionary.* GeoInformation International, New York (USA).

MONTGOMERY, G.; SCHUCH, H., 1993. *GIS Data Conversion Handbook.* GIS World, Inc.; Fort Collins (Colorado, USA).

NARDINOCCHI, C.; FORLANI, G.; ZINGARETTI, P., 2003. Classification and filtering of laser data. ISPRS Workshop on *3D reconstruction from airborne laser scanner and InSAR data.* Dresden (Germany).

NICHOL, D. G., 1990. Region adjacency analysis of remotely-sensed imagery. *International Journal of Remote Sensing*, Vol.11, Nr.11: 2089-2101. Taylor & Francis Ltd.

NUNES, J., 1991. Geographic space as a set of concrete geographical entities. In *Cognitive and Linguistics Aspects of Geographic Space* (MARK D., FRANK A., Eds): 9-33. Kluwer Academic Publishers, Dordrecht.

OEEPE, 2000. 2000 Working Group on laser data acquisition [online]. ISPRS Congress 2000. Available from http://www.geomatic.kth.se/~fotgram/OEEPE/ISPRS_Amsterdam_OEEPE_presentation.pdf [Accessed 15 December 2003].

O'SULLIVAN, D.; TURNER, A., 2001. Visibility graphs and landscape visibility analysis. *International Journal of Geographical Information Science*, Vol.15, No.3: 221-237. Taylor & Francis Ltd.

PENN, A.; CROXFORD, B., 1997. Effects on street grid configuration on kerbside concentrations of vehicular emissions. In Proceedings, *First International Symposium on Space Syntax* (HILLIER B., Ed), 16-18 April: 27.1-27.10. University College London, (England, UK).

PEPONIS, J.; ZIMRING, C.; CHOI, Y., 1990. Finding the building in way-finding. In *Environment and behavior*, Vol. 22: 555-590. Sage Publications.

PEUCKER, T. K.; CHRISMAN, N., 1975. Cartographic data structures. In *The American Cartographer*, Vol.2: 55-69.

RADKE, J.D., 1982. *Pattern Recognition in Circuit Networks*, PhD thesis, Department of Geography, University of British Columbia (USA).

RAPER, J. F.; MAGUIRE, D. J., 1992. Design models and functionality in GIS. In *Computers & Geosciences*, 18: 387-394.

REED, C., 1999. GIS Users Shouldn't Forget About Topology [online]. In *GeoWorld Readers' Forum*. Available from http://www.geoplace.com/gw/1999/0499/499form.asp [Accessed 29 January 2004].

RITCHIE, J., 1996. Remote sensing applications to hydrology: airborne laser altimeters. *Hydrological Sciences Journal*, 41: 625-636.

RIGAUX, J., SCHOLL, M., VOISARD, A., 2002. *Spatial Databases with Applications to GIS*, Chapter 8. Morgan Kaufmann, San Francisco (USA).

ROBERTS, S.A.; HALL, G.B.; CALAMAI, P.H., 2000. Analysing forest fragmentation using spatial autocorrelation, graphs and GIS. *International Journal of Geographical Information Science*, Vol.14, No.2: 185-204. Taylor & Francis Ltd.

SANDWELL, D., 2004. *Use of colour in remote sensing* [online]. Available from http://topex.ucsd.edu/rs/color.pdf [Accessed 17 December 2005].

SEDGEWICK, R., 1988. *Algorithms* (2nd ed.). Addison-Wesley, Reading (MA, USA).

SEDGEWICK, R., 1998. *Algorithms in C, Parts 1-4* (3rd ed.). Chap. 3, 4, 5. Addison-Wesley, Reading (MA, USA).

SEDGEWICK, R., 2002. *Algorithms in C, Part 5* (3rd ed.). Chap. 17, 18. Addison-Wesley, Reading (MA, USA).

SCHALKOFF, R. J., 1989. *Digital Image Processing and Computer Vision*. John Wiley, New York (USA).

SCHALKOFF, R. J., 1992. *Pattern Recognition: Statistical, Structural and Neural Approaches*. John Wiley, New York (USA).

SCHILLING, D.; REVELLE, C.; COHON, J., 1981. An approach to the display and analysis of multiobjective problems. *Socio-Economic Planning Problems*, Vol. 17: 57-63.

SCHUURMAN, N., 2004. *GIS – A short introduction, 4*. Blackwell Publishing Ltd, Bodmin (England, UK).

SITHOLE, G.; VOSSELMAN, G.; 2003. Comparison of filtering algorithms [online]. *Proceedings of ISPRS Working Group III/3 workshop on 3D reconstruction from airborne laser scanner and InSAR data*, 34 (3/W13): 71-78. Dresden (Germany). Available from http://www.isprs.org/commission3/wg3/workshop_laserscanning/papers/Sithole_ALSDD2003.pdf [Accessed 15 December 2003].

SOHN, G., 2004. *Extraction of Buildings from High-Resolution Satellite data and Airborne LiDAR*. PhD thesis (unpublished). Department of Geomatic Engineering, University College London (England, UK).

SOWA, J. F., 2000. *Knowledge Representation - Logical, Philosophical and Computational Foundations*, Chapter 2. Brooks/Cole, Pacific Grove (CA, USA).

STEEL, A. M.; BARNSLEY, M. J.; BARR, S. L., 2003. Inferring urban land use through analysis of the spatial composition of buildings identified in LiDAR and multispectral image data. *Remotely-Sensed Cities* (MESEV, T.V., ed.). Taylor & Francis Ltd., London (England, UK).

SHUFELT, J. A., 1999. Performance evaluation and analysis of monocular building extraction from aerial imagery. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 21, No. 4: 311-326.

SULLIVAN, T. F., 1973. The geometry of solids in Hilbert spaces. *Notre Dame Journal of Formal Logic*, Vol. XIV, No.4: 575-580. NDJFAM.

TARLE, T., 1995. Topology - Why bother? In *GIS '95 Conference Proceedings.* Fort Collins (USA).

TARSKI, A., 1929. Foundations of the geometry of solids. In *Logic, Semantics, Metamathematics* (Papers from 1923 to 1938). Clarendon Press, Oxford (England, UK), 1956: 24-29.

TEGHEM, J.; KUNSCH, P., 1986. A survey of techniques for finding efficient solutions to multiobjective integer linear programming. *Asia Pacific Journal of Operational Research*, Vol. 3: 95-108.

TEMPERLEY, H., 1981. *Graph Theory and Applications.* Ellis Horwood Ltd., Chichester (England, UK).

THEOBALD, D.M, 2001. Topology revisited: representing spatial relations. *International Journal of Geographical Information Science*, Vol. 15, No. 8: 689-705. Taylor & Francis Ltd.

TOUSSAINT, G.T., 1980a. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, Vol.12, No.4: 261-268. Pergamon Press Ltd.

TOUSSAINT, G.T., 1980b. Pattern recognition and geometrical complexity. Proceedings of *V International Conference on Pattern Recognition*: 1324-1347. Miami (USA).

TRANSPORT FOR LONDON (TfL), 2005. History - The Real Underground [online]. Available from http://www.tfl.gov.uk/tube/maps/realunderground/realunderground.html [Accessed 16 July 2005].

TURNER, A.; PENN, A., 1999. Making isovists syntactic: isovist integration analysis (Abstract). In Proceedings, *Second International Symposium on Space Syntax* (HOLANDA F., Ed), 29 March-2 April: pp.08. Universidade de Brasília, Brasília (Brazil).

URQUHART, R., 1982. Graph theoretical clustering based on limited neighbourhood sets. *Pattern Recognition*, Vol.15, No.3: 173-187. Pergamon Press Ltd.

US ARMY CORPS OF ENGINEERS (USACE), 2002. Airborne LiDAR Topographic Surveying. *Engineering and Design – Photogrammetric Mapping*, Chapter 11 [online]. Manual No. 1110-1-1000. Department of Army. Washington, DC (USA). Available from http://www.usace.army.mil/usace-docs/eng-manuals/em1110-1-1000/basic.pdf [Accessed 15 March 2006].

VARZI, A. C., 1996. Reasoning about space: the hole story. In *Logic and Logical Philosophy*, 4: 3-39. Pergamon Press Ltd.

VARZI, A. C., 2003. Mereology. In *Stanford Encyclopaedia of Philosophy* [online]. Available from http://plato.stanford.edu/entries/mereology/ [Accessed 11 October 2005].

WEHR, A.; LOHR U., 1999. Airborne laser scanning – an introduction and overview. *ISPRS Journal of Photogrammetry & Remote Sensing*, 54: 68-82. Elsevier Science Ltd.

WILSON, R., 1996. *Introduction to Graph Theory* (4th ed.). Longman, Harlow (UK).

WOLF, G.W., 1984. A mathematical model of cartographic generalisation. *Geo-processing*, Vol.2, No.3: 271-286.

WORBOYS, M., 1995. *GIS – A Computing Perspective*. Taylor & Francis, London (England, UK).

WORBOYS, M.; DUCKHAM, M., 2004. *GIS – A Computing Perspective* (2nd ed.). CRC Press LLC, Boca Raton (Florida, USA).

ZENG, Yu; ZHANG, J.; WANG, G.; LIN, Z., 2002. Urban land-use classification using integrated airborne laser scanning data and high resolution multi-spectral satellite imagery. *Pecora15/Land Satellite Information IV/ISPRS Commission I/FIEOS 2002Conference* proceedings. Denver (Colorado, USA).

ZHANG, K; CHENG, S.-C.; WHITMAN, D.; SHYU, M.-L.; YAN, J.; ZHANG, C., 2003. A progressive morphological filter for removing non-ground measurements from airborne LiDAR data [online]. In *IEEE Transactions on Geoscience and Remote Sensing*, Vol. 41, No. 4: 872-882. Available from http://ieeexplore.ieee.org/iel5/36/27082/01202973.pdf?isnumber=&arnumber=1202973 [Accessed 20 July 2006].

A AML routines to retrieve polygon adjacencies

adj4graph.aml

```
display 9999
workspace C:\ZP\PhD\Implementations\kew\NE_Zone

/* program to build the graph of polygon adjacencies.

&severity &warning &ignore
&severity &error &routine bailout

&sv oldfull = [show &fullscreen]
&sv oldworkspace = [show &workspace]
&sv oldamlpath = [show &amlpath]

/* closing existing cursors
&s curslist = [show cursors]
&s num = [token %curslist% -count]
&do i = 1 &to %num%
  &s curs2rem = [extract %i% %curslist%]
  cursor %curs2rem% close
  cursor %curs2rem% remove
&end

&messages &off

&sv polycover = arcs45copy
&if [exists %polycover%_adjacencies -info] &then killinfo %polycover%_adjacencies
palinfo %polycover% %polycover%_adjacencies
&if [exists adjacencies -info] &then killinfo adjacencies
pullitems %polycover%_adjacencies adjacencies unit adjacent
&run filteradjacencies.aml   /* vd. code below

ARCPLOT

reselect %polycover% poly
&sv nbrpolys = [extract 2 [show select %polycover% poly]]

reselect adjacencies info
&sv nbradj = [extract 2 [show select adjacencies info]]
aselect adjacencies info

/*create/open the output file
&if [exists outputadj45.dat -file] &then &do
/*      &sv closestat = [close outputadj45.dat]
        &sv delstat = [delete outputadj45.dat]
        &end /*then
&sv fileunit = [open outputadj45.dat openstatus -write]
/*check if the file outputadj45.dat was opened successfully
&type openstatus = %openstatus%
```

```
&type fileunit = %fileunit%
&if %openstatus% <> 0 &then
  &type File outputadj45.dat was not opened successfully!
&else &do
  &type File outputadj45.dat was opened successfully. Writing onto it...
  cursor pointer declare adjacencies info rw
  cursor pointer open
  cursor pointer first
  &do i := 1 &to %nbradj%
    &sv writestat = [write %fileunit% [quote %:pointer.UNIT% %:pointer.ADJACENT%]]
    cursor pointer next
  &end /*end &do i
&end /*end else
cursor pointer close
/*close the output file
&sv closestat = [close %fileunit%]

QUIT

exit
&messages &on

&return
/*
/* ====== routine exit=========
&routine exit
/*&fullscreen %oldfull%
&workspace %oldworkspace%
/*&amlpath %oldamlpath%
&return
/*
/*======routine bailout =======
&routine bailout
&severity &error &fail
&type \--> AML ERROR: %aml$errorfile%
&type --> LINE : %aml$errorline%
&type --> ERROR: %aml$message%
&messages &on
&call exit
&return &error Bailing out of filterdtm.aml
```

filteradjacencies.aml (From Ayugi, 2006)

```
display 9999
workspace C:\ZP\PhD\Implementations\kew\NE_Zone

/* program to interchange column values.

&severity &warning &ignore
&severity &error &routine bailout

&sv oldfull = [show &fullscreen]
&sv oldworkspace = [show &workspace]
&sv oldamlpath = [show &amlpath]

/* closing existing cursors

&s curslist = [show cursors]
&s num = [token %curslist% -count]
&do i = 1 &to %num%
  &s curs2rem = [extract %i% %curslist%]
  cursor %curs2rem% close
  cursor %curs2rem% remove
&end

&messages &off

tables
sel adjacencies
sort unit adjacent
quit

cursor cur1 declare adjacencies info rw
cursor cur1 open
```

```
cursor cur1 first

&sv a1 = %:cur1.unit%
&sv a2 = %:cur1.adjacent%

cursor cur1 next

&do &while %:cur1.aml$next%
        &sv ii = 0
        &do &until %ii% = 1
                &if %a1% = %:cur1.unit% and %a2% = %:cur1.adjacent% &then
                        cursor cur1 delete
                &else &do
                        &sv a1 = %:cur1.unit%
                        &sv a2 = %:cur1.adjacent%
                        cursor cur1 next
                        &sv ii = 1
                &end /* end else
        &end /* ii = 0
&end /* &do &while %:cur1.aml$next%

cursor cur1 close
cursor cur1 remove


exit
&messages &on

&return
/*
/* ====== routine exit==========
&routine exit
/*&fullscreen %oldfull%
&workspace %oldworkspace%
/*&amlpath %oldamlpath%
&return
/*
/*======routine bailout =======
&routine bailout
&severity &error &fail
&type \--> AML ERROR: %aml$errorfile%
&type --> LINE : %aml$errorline%
&type --> ERROR: %aml$message%
&messages &on
&call exit
&return &error Bailing out of filterdtm.aml
```

B Extract from a file of a graph of adjacencies

```
1 2
1 3
1 4
1 5
1 6
1 9
1 13
1 46
1 97
1 165
1 264
1 293
1 302
2 1
2 3
3 1
3 2
3 4
3 5
3 6
3 11
3 12
3 14
3 15
3 30
3 31
3 32
3 34
3 39
3 40
3 42
3 45
3 46
3 51
3 52
3 56
3 57
3 58
3 61
3 63
3 67
3 68
3 70
3 74
3 76
3 77
3 78
3 81
3 84
3 88
3 90
3 94
3 98
```

3 99
3 100
3 106
3 109
3 110
3 112
3 113
3 116
3 117
3 118
3 120
3 121
3 122
3 124
3 125
3 126
3 127
3 129
3 130
3 131
3 132
3 133
3 135
3 137
3 138
3 140
3 141
3 145
3 146
3 147
3 148
3 149
3 150
3 153
3 154
3 155
3 156
3 161
3 165
3 167
3 169
3 171
3 178
3 182
3 185
3 189
3 191
3 195
3 196
3 198
3 201
3 202
3 203
...
[*649 lines*]
...
300 3
301 3
302 1
302 240
303 3
303 305
304 3
304 305
305 241
305 303
305 304

C AML routine to retrieve steep-polygon touchings

nodeshare.aml

```
display 9999
workspace C:\ZP\PhD\Implementations\kew\NE_Zone

/* programme to retrieve polygon nodes lists and
/* to generate a file listing the steep polygons which share a node

&severity &warning &ignore
&severity &error &routine bailout

&sv oldfull = [show &fullscreen]
&sv oldworkspace = [show &workspace]
&sv oldamlpath = [show &amlpath]

/* closing existing cursors
&s curslist = [show cursors]
&s num = [token %curslist% -count]
&do i = 1 &to %num%
  &s curs2rem = [extract %i% %curslist%]
  cursor %curs2rem% close
  cursor %curs2rem% remove
&end

&messages &off

&sv polycover = arcs45copy

/*-----------------------------------------------------------------
&if [exists T1 -info] &then killinfo T1
pullitems %polycover%.aat T1 FNODE# TNODE# LPOLY# RPOLY#

/*              |=========================================|
/*              |                    T1                   |
/*              |=========================================|
/*              |  FNODE#  |  TNODE#  |  LPOLY#  |  RPOLY#  |
/*              |=========================================|

&if [exists T2 -info] &then killinfo T2
TABLES
copy T1 T2
QUIT
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
cursor curT1 declare T1 info rw
cursor curT1 open
cursor curT1 first

cursor curT2 declare T2 info rw
cursor curT2 open
```

```
cursor curT2 first

/* cursor curT1 reads table T1 and inserts the read rows onto T2 in a reversed way
/* in this procedure T1 is an auxiliar table, T2 will contain 2x T1's rows
&do &while %:curT1.aml$next%
        cursor curT2 insert
        &sv :curT2.FNODE# = %:curT1.TNODE#%
        &sv :curT2.TNODE# = %:curT1.FNODE#%
        &sv :curT2.LPOLY# = %:curT1.RPOLY#%
        &sv :curT2.RPOLY# = %:curT1.LPOLY#%
        cursor curT1 next
&end /*do-while

cursor curT1 close
cursor curT1 remove
cursor curT2 close
cursor curT2 remove

/*              |=========================================|
/*              |                  T2                     |
/*              |=========================================|
/*              |  FNODE#  |  TNODE#  |  LPOLY#  |  RPOLY#  |
/*              |=========================================|
/*              |    a     |    b     |    c     |    d     |
/*              |=========================================|
/*              |    b     |    a     |    d     |    c     |
/*                 ...         ...        ...        ...
/*              |=========================================|

&if [exists T3 -info] &then killinfo T3
pullitems T2 T3 FNODE# LPOLY#

/*              |====================|
/*              |         T3         |
/*              |====================|
/*              |  FNODE#  | LPOLY#  |
/*              |====================|
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
/* procedure to sort table T3 and delete repeated rows
TABLES
sel T3
sort FNODE# LPOLY#
QUIT

cursor cur1 declare T3 info rw
cursor cur1 open
cursor cur1 first

&sv a1 = %:cur1.FNODE#%
&sv a2 = %:cur1.LPOLY#%

cursor cur1 next

&do &while %:cur1.aml$next%
        &sv ii = 0
        &do &until %ii% = 1
                &if %a1% = %:cur1.FNODE#% and %a2% = %:cur1.LPOLY#% &then
                        cursor cur1 delete
                &else &do
                        &sv a1 = %:cur1.FNODE#%
                        &sv a2 = %:cur1.LPOLY#%
                        cursor cur1 next
                        &sv ii = 1
                &end /* else
        &end /* ii = 0
&end /* &do &while %:cur1.aml$next%

cursor cur1 close
cursor cur1 remove
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
&if [exists T4 -info] &then killinfo T4
pullitems %polycover%.pat T4 %polycover%# SLOPECLASS
```

```
/*              |================================|
/*              |               T4               |
/*              |================================|
/*              |  %polycover%#  |  SLOPECLASS   |
/*              |================================|

TABLES
sel T4
alter %polycover%# PolyID ,,,,,,

/*              |==========================|
/*              |            T4            |
/*              |==========================|
/*              |  PolyID  |  SLOPECLASS   |
/*              |==========================|

asel
sel T3
alter FNODE# NODE# ,,,,,,
alter LPOLY# PolyID ,,,,,,

/*              |======================|
/*              |          T3          |
/*              |======================|
/*              |   NODE#  |   PolyID  |
/*              |======================|
QUIT

&if [exists T5 -info] &then killinfo T5
joinitem T3 T4 T5 PolyID

/*              |====================================|
/*              |                T5                  |
/*              |====================================|
/*              |   NODE#  |   PolyID  | SLOPECLASS  |
/*              |====================================|

TABLES
sel T5
resel SLOPECLASS = 0
purge
yes
QUIT

/*              |===================================|
/*              |                T5                 |
/*              |===================================|
/*              |   NODE#  |  PolyID  | SLOPECLASS  |
/*              |===================================|
/*              |    ...   |   ...    |     1       |
/*              |===================================|


dropitem T5 T5 SLOPECLASS

/*              |====================|
/*              |         T5         |
/*              |====================|
/*              |   NODE#  |  PolyID |
/*              |====================|

&if [exists T5_2 -info] &then killinfo T5_2
TABLES
copy T5 T5_2

/*              |====================|
/*              |        T5_2        |
/*              |====================|
/*              | NODE# |  PolyID_2  |
/*              |====================|

sel T5_2
alter PolyID PolyID_2 ,,,,,,
sort NODE# PolyID_2
```

```
sel T5
sort PolyID NODE#
QUIT

ARCPLOT
reselect T5_2 info
&sv lastrecord = [extract 2 [show select T5_2 info]]
aselect T5_2 info
QUIT

&if [exists T6 -info] &then killinfo T6
joinitem T5 T5_2 T6 NODE#

/*              |=================================|
/*              |               T6                |
/*              |=================================|
/*              |  NODE#  |   PolyID   | PolyID_2 |
/*              |=================================|

TABLES
sel T6
calc PolyID = -9999
QUIT
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
/* previous "joinitem" was just for creation purposes of T6
/* now we are going to make sure that all occurrences from both
/* T5 and T5_2 are summarized in T6 and the "join" is properly done

cursor curT5 declare T5 info rw
cursor curT5 open
cursor curT5 first

cursor curT6 declare T6 info rw
cursor curT6 open
cursor curT6 first

/* cursor curT5 reads table T5, row by row,
/* and joins it with corresponding rows in table T5_2 (joining item: NODE#);
/* the result is inserted by curT6 into T6
&sv cont1 = %:curT5.aml$next%
&sv countT5rows = 0
&sv ii = 0

&do &while %:curT5.aml$next%
    &do &until %ii% - 1 = 0

        &sv countT5rows = %countT5rows% + 1
        /*&type ---------------  ROW Nr %countT5rows% READ IN TABLE T5 ---------------
        &sv jj = 0
        &sv kk = 0

        cursor curT5_2 declare T5_2 info rw NODE# = %:curT5.NODE#%
        cursor curT5_2 open
        &sv cont2 = %:curT5_2.aml$next%
        cursor curT5_2 first

        &do &while %:curT5_2.aml$next%
                &do &until %jj% - 1 = 0
                        cursor curT6 insert
                        &sv :curT6.NODE# = %:curT5.NODE#%
                        &sv :curT6.PolyID = %:curT5.PolyID%
                        &sv :curT6.PolyID_2 = %:curT5_2.PolyID_2%
                        cursor curT5_2 next
                        &if %cont2% <> %:curT5_2.aml$next% &then &sv jj = 1
                &end /*do-until
        &end /*do-while

        cursor curT5_2 close
        cursor curT5_2 remove

        cursor curT5 next
        &if %cont1% <> %:curT5.aml$next% &then &sv ii = 1

    &end /*do-until
```

```
&end /*do-while

cursor curT5 close
cursor curT5 remove
cursor curT6 close
cursor curT6 remove
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
/* rows referring to polygons which share nodes only with other
/* different polygons are left in T6, all the others are deleted
TABLES
sel T6
resel PolyID = -9999
purge
yes
asel
resel PolyID = PolyID_2
purge
yes
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
/* procedure to sort table T6 and delete repeated rows (Ayugi, 2006)
asel
sel T6
sort PolyID PolyID_2
QUIT

cursor cur1 declare T6 info rw
cursor cur1 open
cursor cur1 first

&sv a1 = %:cur1.PolyID%
&sv a2 = %:cur1.PolyID_2%

cursor cur1 next

&do &while %:cur1.aml$next%
        &sv ii = 0
        &do &until %ii% = 1
                &if %a1% = %:cur1.PolyID% and %a2% = %:cur1.PolyID_2% &then
                        cursor cur1 delete
                &else &do
                        &sv a1 = %:cur1.PolyID%
                        &sv a2 = %:cur1.PolyID_2%
                        cursor cur1 next
                        &sv ii = 1
                &end /* else
        &end &do &until
&end /* &do &while

cursor cur1 close
cursor cur1 remove
/*-----------------------------------------------------------------

/*-----------------------------------------------------------------
/*create/open the output file
ARCPLOT
reselect T6 info
&sv lastrecord = [extract 2 [show select T6 info]]
aselect T6 info
QUIT

&if [exists outputnodeshare.dat -file] &then &do
/*      &sv closestat = [close outputnodeshare.dat]
        &sv delstat = [delete outputnodeshare.dat]
&end /*then
&sv fileunit = [open outputnodeshare.dat openstatus -write]
/*check if the file outputnodeshare.dat was opened successfully
&type openstatus = %openstatus%
&type fileunit = %fileunit%
&if %openstatus% <> 0 &then
  &type File outputnodeshare.dat was not opened successfully!
&else &do
  &type File outputnodeshare.dat was opened successfully. Writing into it...
```

```
  cursor pointer declare T6 info rw
  cursor pointer open
  cursor pointer first
  &do i := 1 &to %lastrecord%
       &if %:pointer.POLYID_2% > %:pointer.POLYID% &then
            &sv   writestat   =   [write   %fileunit%   [quote   %:pointer.POLYID%
%:pointer.POLYID_2%]]
       cursor pointer next
  &end /*do i
&end /*else
cursor pointer close
cursor pointer remove
/*close the output file
&sv closestat = [close %fileunit%]
/*-----------------------------------------------------------------

exit
&messages &on

&return
/*
/* ====== routine exit=========
&routine exit
/*&fullscreen %oldfull%
&workspace %oldworkspace%
/*&amlpath %oldamlpath%
&return
/*
/*======routine bailout =======
&routine bailout
&severity &error &fail
&type \--> AML ERROR: %aml$errorfile%
&type --> LINE : %aml$errorline%
&type --> ERROR: %aml$message%
&messages &on
&call exit
&return &error Bailing out of filterdtm.aml
```

D Example of a file of a graph of steep-polygon touchings

2 4
2 6
4 5
4 34
5 11
5 12
5 13
5 24
5 30
5 31
5 60
5 66
5 82
5 92
5 97
5 101
5 103
5 125
11 12
12 30
12 33
12 38
12 57
14 15
15 56
15 63
15 78
15 129
15 136
15 155
15 172
15 191
15 205
15 212
24 30
30 38
30 61
34 40
39 45
40 58
46 112
46 189
46 209
46 223
46 239
46 241
46 258

46 274
46 282
46 284
46 293
46 296
46 302
51 52
60 66
61 77
68 77
74 76
92 101
97 165
102 104
113 116
116 117
121 124
121 126
125 137
125 182
129 130
135 138
138 141
140 141
141 145
141 146
146 147
159 163
160 168
162 166
165 182
171 182
171 213
172 177
172 191
191 202
191 205
193 194
211 215
223 226
226 233
228 231
233 239
233 241
234 245
235 278
239 241
241 274
241 282
241 287
241 302
241 303
241 304
244 262
248 251
250 256
250 260
251 255
256 260
269 272
291 295
293 302
295 301
297 304
299 300
301 303
303 304

E The spatial topology analysis program

Item.h *(header file)*

```
#include <stdlib.h>

typedef int Item;
```

QUEUE.h *(header file)*

```
#include "Item.h"

void QUEUEinit(int);
int QUEUEempty();
void QUEUEput(Item);
int QUEUEget();
```

SceneAnalysis.c

```
** This program does:
**
**      - Reads a set of graph edges from an input file and builds the respective
**        adjacency-list representation of the given graph;
**
**      - If chosen by the user, it detects all cycles in the graph;
**
**      - Starting from a given vertex, traverses the whole graph, either by using
**        Depth-first Search algorithm or Breadth-first Search (recommended for urban
scene
**        analysis), and determines how many levels of depth each vertex is away from
the root;
**
**        Given any vertex as the root, the resulting spanning trees of the graph
traversal
**        are sent to output files (either simple sequences of tree edges or with
comments);
**
**      - It analyzes the spatial topology: based upon the Breadth-first search
algorithm,
**       it traverses the whole graph identifying potential containment units, and
also detecting
**       different levels of containment (which polygon contains which) within a
given unit -
**       - the Containment-first search;
```

```
**
**       Translates previous results of the spatial topology analysis into colours.
The steep & flat regions
**       in the original map will be mapped according to: their containment unit
(each one is assigned a
**              different colour),and also their level of containment within the
respective unit
**       (each level is assigned a different tone of the respective unit's colour);
**
**     - Based on a few analytical rules, it performs a visual generalisation of the
map by
**       updating the colours of steep-polygon islands within flat polygons.

**
** Created:  JPdeA, 18/11/2005
** Modified: JPdeA, 12/5/2006
*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <stddef.h>
#include <string.h>
#include <math.h>
#include "Item.h"
#include "QUEUE.h"

#define LEVELSPACES 500

typedef struct node *a_link;
struct node
{
       int v;
       a_link next;
};

typedef struct colour node_colours;
struct colour
{
       int containment_level;
       int containment_unit;
       int valence;
       int parent;
       int FillStyle;
       int H;
       int S;
       int V;
};

static a_link head, tail;

/* FUNCTION PROTOTYPES */
a_link NEW(int, a_link);
int INIT_adj(int, int*, a_link*);
int INIT_touch(int, int*, a_link*);
void visit(int);
void depth_first_search(int, char*, int, char*, int*, int, a_link*, int, FILE*,
char);
void breadth_first_search(int, char*, int, char*, int*, int, a_link*, int, FILE*,
char);
char* create_spantree_file(int, int, char);
char* create_cycles_file(int, int);
char* create_CFStree_file(int, int);
char* create_CFSRationalization_file(int, int);
void CycleDetection_DFS(int, int, char*, int, int*, int, a_link*, int, FILE*);
void containment_first_search(int, char*, int, a_link*, a_link*, node_colours*, int,
FILE*, char);
void PolyRing_containment(int, int, char*, a_link*, node_colours*, int);
void colouring(int, int, node_colours*);
void rationalization(int, a_link*, node_colours*, FILE*);

/* IMPLEMENTATION OF FUNTIONS */

/* function to create a new link in the graph linked list */
a_link NEW(int v, a_link next)
{
```

```
        a_link x = (struct node *) malloc(sizeof (*x));
        x->v = v;
        x->next = next;
        return x;
} /*NEW*/

/* implementation of the 4 following functions to create a FIFO queue linked list */
void QUEUEinit(int maxN)
{
        head = NULL;
} /*QUEUEinit*/

int QUEUEempty()
{
        return head == NULL;
} /*QUEUEempty*/

void QUEUEput(Item v)
{
        if (head == NULL)
        {
                head = (tail = NEW(v, head));
                return;
        }
        tail->next = NEW(v, tail->next);
        tail = tail->next;
} /*QUEUEput*/

Item QUEUEget()
{
        Item v = head->v;
        a_link t = head->next;
        free(head);
        head = t;
        return v;
} /*QUEUEget*/

/* Function 1, to read the adjacency graph edges, and to build its respective
adjacency-lists representation */
int INIT_adj(int nbrpolys, int *nbredges, a_link *adj)
{
        int i, j, countnbredges;
        FILE *ifp;

        /* initialization of the array of lists */
        for (i = 0; i < nbrpolys; i++)
                adj[i] = NULL;

        /* opens the input file of adjacencies */
        if ((ifp = fopen("c:/zp/phd/implementations/malta/outputadj20.dat", "r")) ==
NULL)
        {
                printf("Couldn't open the input file of adjacencies!\n");
                return 1;
        }
        else printf("Input file of adjacencies opened successfully!\n");

        countnbredges = *nbredges;
        /* reads the input file and builds the adjacency-list */
        while (!feof(ifp))
        {
                if (fscanf(ifp,"%d %d\n", &i, &j) != 2)
                {
                        if(!feof(ifp))
                        {
                                printf("\n\nError       reading       %s       file\n",
"outputadj30.dat");
                                fclose(ifp);
                                return 2;
                        }
                }
                else
                {
                        i--;
                        j--;
                        if (j > i)
```

```
                        {
                                countnbredges++;
                                adj[j] = NEW(i, adj[j]);
                                adj[i] = NEW(j, adj[i]);
                        }
                }
        } /* while */
        *nbredges = countnbredges;
        fclose(ifp);
        return 0;
} /*INIT_adj*/

/* Function 2, to read the graph with the touching relationships between steep
polygons,
** and to build its respective "touching-list representation" */
int INIT_touch(int nbrpolys, int *nbrtouchingedges, a_link *touch)
{
        int i, j, countnbredges;
        FILE *ifp;

        /* initialization of the array of lists */
        for (i = 0; i < nbrpolys; i++)
                touch[i] = NULL;

        /* opens the input file of touchings */
        if   ((ifp   =   fopen("c:/zp/phd/implementations/malta/outputnodeshare20.dat",
"r")) == NULL)
        {
                printf("Couldn't open the input file of touchings!\n");
                return 3;
        }
        else printf("Input file of touchings opened successfully!\n");

        countnbredges = *nbrtouchingedges;
        /* reads the input file and builds the touchings-list */
        while (!feof(ifp))
        {
                if (fscanf(ifp,"%d %d\n", &i, &j) != 2)
                {
                        if(!feof(ifp))
                        {
                                printf("\n\nError       reading       %s       file\n",
"outputnodeshare20.dat");
                                fclose(ifp);
                                return 4;
                        }
                }
                else
                {
                        i--;
                        j--;
                        if (j > i)
                        {
                                countnbredges++;
                                touch[j] = NEW(i, touch[j]);
                                touch[i] = NEW(j, touch[i]);
                        }
                }
        } /* while */
        *nbrtouchingedges = countnbredges;
        fclose(ifp);
        return 0;
} /*INIT_touch*/

/* Function 3, to create output file to write a spanning tree in */
char* create_spantree_file(int i, int nbrpolys, char option2)
{
        char *filename_head, *filename_num, *filename_tail, *totalfinalname;
        int n;

        if (option2 == 'D' || option2 == 'd')
                filename_head                                                    =
"e:/zp/phd2/implementations/treefiles/malta/20DegreeSlope/depth_first_search/outputtr
ee_";
        else
```

```
                filename_head                                                    =
"e:/zp/phd2/implementations/treefiles/malta/20DegreeSlope/breadth_first_search/output
tree_";

        filename_tail = ".dat";

        /* allocation of memory for 'filename_num' & assignment of first value */
        filename_num = malloc(nbrpolys);
        sprintf(filename_num, "%05d", i + 1);

        /* allocation of memory & initialisation of the string 'totalfinalname' */
        n = strlen(filename_head) + strlen(filename_num) + strlen(filename_tail) + 1;
        totalfinalname = malloc(n*sizeof(char));
        totalfinalname[0] = '\0';

        /* composition of final string for 'totalfinalname' */
        strcat(totalfinalname, filename_head);
        strcat(totalfinalname, filename_num);
        strcat(totalfinalname, filename_tail);

        free(filename_num);

        return totalfinalname;
} /*create_spantree_file*/

/* Function 4, to recursively traverse the graph by visiting all its vertices
** and determining their level of adjacency, starting from any vertex:
** Depth-first Search algorithm is used */
void depth_first_search(int k, char *visited, int level, char *wspc, int *levels, int
start_node, a_link *adj, int nbrpolys, FILE *ofp, char option1)
{
        int i;
        a_link t;

        visited[k] = 1; /* vertex k is being visited */
        level++; /*counts the level of depth of the tree*/

        if (option1 == 'C' || option1 == 'c')
        {
                if (level > LEVELSPACES)
                        printf("Too many levels! WSPC will be assigned %d spaces.\n",
LEVELSPACES);
                else
                        strcat(wspc, " "); /*compose a string 'wspc' containing 'level'
spaces*/
        }

        *(levels + start_node*nbrpolys + k) = level;/*matrix to store the levels of
adjacency of each vertex from a given root*/

        for (t = adj[k]; t != NULL; t = t->next)
        {
                if (!visited[t->v])
                {
                        if (option1 == 'C' || option1 == 'c')
                        {       /* writes on the output commented file the new depth
level */
                                fprintf(ofp, "Level %d of containment\n", level);
                                /* writes on the output commented file a new tree
traversal edge */
                                fprintf(ofp, "%s%d %d\n", wspc, k + 1, t->v + 1);
                        }
                        else  /* writes on the output text data file a new tree
traversal edge */
                                fprintf(ofp, "%d %d\n", k + 1, t->v + 1);

                        depth_first_search(t->v,   visited,   level,   wspc,   levels,
start_node, adj, nbrpolys, ofp, option1);
                }
        } /*for*/

        if (option1 == 'C' || option1 == 'c')
        {
                fprintf(ofp, "Level %d done\n", level);
                wspc[strlen(wspc) - 1] = '\0';
        }
```

```
} /*depth_first_search*/

/* Function 5, to traverse the graph by visiting all its vertices
** and determining their level of adjacency, starting from any vertex:
** Breadth-first Search algorithm is used */
void breadth_first_search(int k, char *visited, int level, char *wspc, int *levels,
int start_node, a_link *adj, int nbrpolys, FILE *ofp, char option1)
{
        int i;
        a_link t;
        char *marked;

        QUEUEinit(nbrpolys);
        QUEUEput(k);

        /* dynamic allocation of memory & initialisation of "marked" */
        marked = (char *) malloc(nbrpolys*sizeof (char));
        for (i = 0; i < nbrpolys; i++)
                        marked[i] = 0;

        level++; /*counts the level of adjacency*/
        marked[k] = 1;

        while (!QUEUEempty())
        {
                if (visited[k = QUEUEget()] == 0)
                {
                        visited[k] = 1; /*indicates that vertex "k" is being visited*/
                        level++; /*counts the level of adjacency*/

                        if (option1 == 'C' || option1 == 'c')
                        {
                                if (level > LEVELSPACES)
                                        printf("Too many levels! WSPC will be assigned
%d spaces.\n", LEVELSPACES);
                                else
                                        strcat(wspc,  "  "); /*compose a string 'wspc'
containing 'level' spaces*/
                        }

                        *(levels + start_node*nbrpolys + k) = level; /*matrix to store
the levels of adjacency of each vertex from a given root*/

                        /* visiting "k" by checking all its adjacencies */
                        for (t = adj[k]; t != NULL; t = t->next)
                        {
                                if (visited[t->v] == 0)
                                {
                                        /* if from-node "k" is being visited and to-node
"t->v" hasn't been seen yet from any other node then... */
                                        if (marked[k] == 1 && marked[t->v] == 0)
                                        {
                                                if (option1 == 'C' || option1 == 'c')
                                                {       /* writes on the output Commented
File the new depth level */
                                                        /*fprintf(ofp,   "Level   %d   of
containment\n", level);*/
                                                        /* writes on the output Commented
File a new tree traversal edge */
                                                        fprintf(ofp, "%s%d %d\n", wspc, k
+ 1, t->v + 1);
                                                }
                                                else /* writes on the output Text Data
File a new tree traversal edge */
                                                        fprintf(ofp, "%d %d\n", k + 1, t-
>v + 1);
                                        }

                                        /* although not visited yet, "t->v" has already
been seen from "k" */
                                        marked[t->v] = 1;
                                        QUEUEput(t->v);
                                } /*if*/
                        } /*for*/

                        if (option1 == 'C' || option1 == 'c')
```

```
                        {
                                /*fprintf(ofp, "Level %d done\n", level);*/
                                wspc[strlen(wspc) - 1] = '\0';
                        }
                } /*top if*/
        } /*while*/
} /*breadth_first_search*/

/* Function 6, to create output file for the cycle detection results */
char* create_cycles_file(int i, int nbrpolys)
{
        char *filename_head, *filename_num, *filename_tail, *totalfinalname;
        int n;

        filename_head                                                                    =
"e:/zp/phd2/implementations/treefiles/malta/20DegreeSlope/cycles_detection/cycles_";
        filename_tail = ".dat";

        /* allocation of memory for 'filename_num' & assignment of first value */
        filename_num = malloc(nbrpolys);
        sprintf(filename_num, "%05d", i + 1);

        /* allocation of memory & initialisation of the string 'totalfinalname' */
        n = strlen(filename_head) + strlen(filename_num) + strlen(filename_tail) + 1;
        totalfinalname = malloc(n*sizeof(char));
        totalfinalname[0] = '\0';

        /* composition of final string for 'totalfinalname' */
        strcat(totalfinalname, filename_head);
        strcat(totalfinalname, filename_num);
        strcat(totalfinalname, filename_tail);

        free(filename_num);

        return totalfinalname;
} /*create_cycles_file*/

/* Function 7, based on the Depth-first Search algorithm to
** recursively traverse the whole graph detecting cycles */
void CycleDetection_DFS(int parent, int k, char *visited, int level, int *levels, int
start_node, a_link *adj, int nbrpolys, FILE *ofp)
{
        int i;
        a_link t;

        visited[k] = 1; /* vertex k is being visited */
        level++; /*counts the level of depth of the tree*/

        *(levels + start_node*nbrpolys + k) = level;/*matrix to store the levels of
adjacency of each vertex from a given root*/
        for (t = adj[k]; t != NULL; t = t->next)
        {
                if (!visited[t->v])
                {
                        fprintf(ofp, "%d %d\n", k + 1, t->v + 1);
                        CycleDetection_DFS(k, t->v, visited, level, levels, start_node,
adj, nbrpolys, ofp);
                }
                else
                        if (t->v == parent)
                        {
                                fprintf(ofp, "Pointing back to parent: %d %d\n", k + 1,
t->v + 1);
                        }
                        else /* (t->v != parent) */
                        {
                                fprintf(ofp, "Cycle detected: %d %d\n", k + 1, t->v +
1);
                        }
        } /*for*/
} /*CycleDetection_DFS*/

/* Function 8, to create output file for the CFS tree, with the levels of containment
*/
char* create_CFStree_file(int i, int nbrpolys)
{
```

```
        char *filename_head, *filename_num, *filename_tail, *totalfinalname;
        int n;

        filename_head                                                          =
"e:/zp/phd2/implementations/treefiles/malta/20DegreeSlope/containment_first_search/CF
Stree_";
        filename_tail = ".dat";

        /* allocation of memory for 'filename_num' & assignment of first value */
        filename_num = malloc(nbrpolys);
        sprintf(filename_num, "%05d", i + 1);

        /* allocation of memory & initialisation of the string 'totalfinalname' */
        n = strlen(filename_head) + strlen(filename_num) + strlen(filename_tail) + 1;
        totalfinalname = malloc(n*sizeof(char));
        totalfinalname[0] = '\0';

        /* composition of final string for 'totalfinalname' */
        strcat(totalfinalname, filename_head);
        strcat(totalfinalname, filename_num);
        strcat(totalfinalname, filename_tail);

        free(filename_num);

        return totalfinalname;
} /*create_CFStree_file*/

/* Function 9, to create output file for the results of the urban topology analysis
*/
char* create_CFSRationalization_file(int i, int nbrpolys)
{
        char *filename_head, *filename_num, *filename_tail, *totalfinalname;
        int n;

        filename_head                                                          =
"e:/zp/phd2/implementations/treefiles/malta/20DegreeSlope/CFS&Rationalization/CFS&Rat
ionalization_";
        filename_tail = ".dat";

        /* allocation of memory for 'filename_num' & assignment of first value */
        filename_num = malloc(nbrpolys);
        sprintf(filename_num, "%05d", i + 1);

        /* allocation of memory & initialisation of the string 'totalfinalname' */
        n = strlen(filename_head) + strlen(filename_num) + strlen(filename_tail) + 1;
        totalfinalname = malloc(n*sizeof(char));
        totalfinalname[0] = '\0';

        /* composition of final string for 'totalfinalname' */
        strcat(totalfinalname, filename_head);
        strcat(totalfinalname, filename_num);
        strcat(totalfinalname, filename_tail);

        free(filename_num);

        return totalfinalname;
} /*create_CFSRationalization_file*/

/* Function 10, based on the Breadth-first search algorithm to traverse the whole
graph
** detecting different levels of containment (which polygons contain which) */
void containment_first_search(int root, char *visited, int start_node, a_link *adj,
a_link *touch, node_colours *polyclr, int nbrpolys, FILE *ofp, char option2)
{
        int i, k, previous, countValence, countContainmentUnits, countNewUrbanSets;
        a_link t;
        char *marked, *checked;

        /* dynamic allocation of memory & initialisation of arrays: marked and checked
*/
        marked = (char *) malloc(nbrpolys*sizeof (char));
        for (i = 0; i < nbrpolys; i++)
                marked[i] = 0;
        marked[root] = 1;

        checked = (char *) malloc(nbrpolys*sizeof (char));
```

```
        for (i = 0; i < nbrpolys; i++)
                checked[i] = 0;

        /* from vertex 'root' all the others are tagged as "not visited" yet -> 0 */
        for (i = 0; i < nbrpolys; i++)
                visited[i] = 0;

        polyclr[root].containment_level = 0; /* the level of containment of the tree
root */
        polyclr[root].containment_unit = 0; /* the tree root doesn't belong to any
particular urban group */
        countContainmentUnits = 0;

        QUEUEinit(nbrpolys);
        QUEUEput(root);
        /* takes the Queue and starts going through its elements */
        while (!QUEUEempty())
        {
                if (visited[k = QUEUEget()] == 0) /*takes the front vertex in the
Queue */
                {
                        if (k == root)
                                fprintf(ofp,     "Level     of     containment     %d\n",
polyclr[root].containment_level);
                        else
                                if            (polyclr[k].containment_level            >
polyclr[previous].containment_level)
                                        fprintf(ofp,   "Level   of   containment   %d\n",
polyclr[k].containment_level);

                        visited[k] = 1;/*indicates that vertex "k" is being visited*/

                        /* starts counting valence of vertex "k" */
                        if (k != root)
                                countValence = 0;

                        /* visiting "k" by checking all its adjacencies */
                        for (t = adj[k]; t != NULL; t = t->next)
                        {
                                if (k != root)
                                        countValence++;

                                if (visited[t->v] == 0)
                                {
                                        if (k == root)
                                        {
                                                /* writes on the output Text Data File a
new tree traversal edge */
                                                fprintf(ofp, "%d %d\n", root + 1, t->v +
1);
                                                /* although not visited yet, "t->v" has
already been seen either from "root" or a previous "k" */
                                                marked[t->v] = 1;

                                                polyclr[t->v].parent = root;
                                                polyclr[t->v].containment_level         =
polyclr[root].containment_level + 1; /*1st level of containment*/

                                                /* Each root's adjacency may correspond
to the existence of a new urban-block */
                                                /* If it's the case: */
                                                if (polyclr[t->v].containment_unit = -9)
                                                {
                                                        countContainmentUnits++;
                                                        /* steep t->v belongs to the new
detected urban-block...*/
                                                        polyclr[t->v].containment_unit  =
countContainmentUnits;
                                                        /*... as well as the respective
connected bit of the graph of touchings */
                                                        PolyRing_containment(t->v,
nbrpolys, checked, touch, polyclr, countContainmentUnits);
                                                } /*if*/
                                        }
                                        else
```

```
                                                /* if from-node "k" is being visited and
to-node "t->v" hasn't been seen yet from any other node then... */
                                                if (marked[k] == 1 && marked[t->v] == 0)
                                                {
                                                        /* writes on the output Text Data
File a new tree traversal edge */
                                                        fprintf(ofp, "%d %d\n", k + 1, t-
>v + 1);
                                                        polyclr[t->v].parent = k;
                                                        /* although not visited yet, "t-
>v" has already been seen either from "root" or a previous "k" */
                                                        marked[t->v] = 1;
                                                        polyclr[t->v].containment_level =
polyclr[k].containment_level + 1;
                                                        polyclr[t->v].containment_unit  =
polyclr[k].containment_unit;
                                                }
                                        QUEUEput(t->v);
                                } /*if*/
                        } /*for*/
                        previous = k;
                        polyclr[k].valence = countValence;
                } /*top 'if'*/
        } /*while*/

        polyclr[root].valence = countContainmentUnits; /*in reality it's not the
root's valence, but the nbr of potential urban-blocks contained by the root*/

        /*if the number of urban sets is greater than 12 then a rearrangement is
needed in order to
        **optimize the use of the colour gamut; thus, if it's the case, steep islands
within
        **the root polygon are discarded as potential urban sets and no colour will be
assigned to them*/
        if (polyclr[root].valence > 12)
        {
                for (i = 0; i < nbrpolys; i++)
                        marked[i] = 0;
                marked[root] = 1;
                for (i = 0; i < nbrpolys; i++)
                        checked[i] = 0;
                for (i = 0; i < nbrpolys; i++)
                        visited[i] = 0;

                countNewUrbanSets = 0;

                QUEUEinit(nbrpolys);
                QUEUEput(root);
                while (!QUEUEempty())
                {
                        if (visited[k = QUEUEget()] == 0)
                        {
                                visited[k] = 1;

                                for (t = adj[k]; t != NULL; t = t->next)
                                {
                                        if (visited[t->v] == 0)
                                        {
                                                if (k == root)
                                                {
                                                        marked[t->v] = 1;
                                                        if ((polyclr[t->v].valence != 1)
&& (checked[t->v] == 0))
                                                        {
                                                                countNewUrbanSets++;
                                                                polyclr[t-
>v].containment_unit = countNewUrbanSets; /*n non-single valent vertices adjacent to
the root determine n potential urban blocks */
                                                                PolyRing_containment(t-
>v, nbrpolys, checked, touch, polyclr, countNewUrbanSets);
                                                        } /*if*/
                                                        else
                                                                if (polyclr[t->v].valence
== 1)
                                                                {
                                                                        checked[t->v] = 1;
```

```
                                                                    polyclr[t-
>v].containment_unit = -9; /*single valent polygons will be cleared out */
                                                                    }
                                                    }
                                                    else
                                                            /* if from-node "k" is being
visited and to-node "t->v" hasn't been seen yet from any other node then... */
                                                            if (marked[k] == 1 && marked[t-
>v] == 0)
                                                            {
                                                                    marked[t->v] = 1;
                                                                    polyclr[t-
>v].containment_unit = polyclr[polyclr[t->v].parent].containment_unit;
                                                            }
                                                    QUEUEput(t->v);
                                            } /*if*/
                                    } /*for*/
                            } /*top 'if'*/
                    } /*while*/

                    polyclr[root].valence = countNewUrbanSets;

            } /*if polyclr[root].valence>12*/

            colouring(root, nbrpolys, polyclr);

} /*containment_first_search*/

/* Function 11, to recursively traverse the graph of touchings (of steep polygons),
** starting from a given "steep" vertex, and to determin which potential urban-set
** correponds to which connected sub-graph; Depth-first search algorithm is used */
void PolyRing_containment(int k, int nbrpolys, char *checked, a_link *touch,
node_colours *polyclr, int countContainmentUnits)
{
        a_link u;

        checked[k] = 1; /* steep vertex "k" is being checked */

        for (u = touch[k]; u != NULL; u = u->next)
        {
                if (checked[u->v] == 0)
                {
                        polyclr[u->v].containment_unit = countContainmentUnits;
                        PolyRing_containment(u->v, nbrpolys, checked, touch, polyclr,
countContainmentUnits);
                }
        } /*for*/
} /*PolyRing_containment*/

/* Function 12, to colour map polygons according to
** the corresponding vertices' attributes */
void colouring(int root, int nbrpolys, node_colours *polyclr)
{
        int i, j, *nbrLevels_UrbSet;
        int  dH, *dSV;

        /* dynamic allocation of memory & initialisation of the arrays:
nbrLevels_UrbSet */
        nbrLevels_UrbSet = (int *) malloc(polyclr[root].valence*sizeof (int));
        for (j = 0; j <= polyclr[root].valence; j++)
                        nbrLevels_UrbSet[j] = -9;

        /* calculates the maximum number of levels of containment per containment_unit
*/
        for (i = 0; i < nbrpolys; i++)
        {
                if                      (polyclr[i].containment_level                   >
nbrLevels_UrbSet[polyclr[i].containment_unit])
                        nbrLevels_UrbSet[polyclr[i].containment_unit]                   =
polyclr[i].containment_level;
        }

        /* calculation of the increments dH (to assign a different colour to each
urban-block); and
        ** dSV (to set a gradient of tones of a given colour for the different levels
of containment
```

```
        ** within a given urban-block) */
        dH = (int) (floor(360 / polyclr[root].valence));

        dSV = (int *) malloc(polyclr[root].valence*sizeof (int));
        for (j = 1; j <= polyclr[root].valence; j++)
                        dSV[j] = (int) (floor(130 / nbrLevels_UrbSet[j]));
        dSV[0] = 0;

        /* setting the colour & fill style for every map polygon according to
        ** the level of containment & urban-block that it belongs to*/
        for (i = 0; i < nbrpolys; i++)
        {
                /* setting the fill style */
                if ((polyclr[i].containment_level%2) != 0)
                        polyclr[i].FillStyle = 1; /*hashed pattern style for steep
polygons*/
                else
                        polyclr[i].FillStyle = 0; /*solid colour style for flat
polygons*/

                if (i == root) /* root's corresponding polygon is mapped in "white" */
                {
                        polyclr[i].H = 60;
                        polyclr[i].S = 0;
                        polyclr[i].V = 100;
                }
                else
                {
                        polyclr[i].H = dH*polyclr[i].containment_unit; /*colour depends
on the urban-set*/
                        polyclr[i].S                                                =
dSV[polyclr[i].containment_unit]*polyclr[i].containment_level; /*colour tone depends
on the level of containment*/
                        if (polyclr[i].S <= 100)
                                polyclr[i].V = 100;
                        else
                        {
                                polyclr[i].V = 200 - polyclr[i].S;
                                polyclr[i].S = 100;
                        } /*else*/
                } /*else*/
        } /*for*/
} /*colouring*/

/* Function 13, to rationalize some of the results of the urban topology analysis
** and to write in the output file the final results */
void rationalization(int nbrpolys, a_link *adj, node_colours *polyclr, FILE *ofp)
{
        int i;
        a_link t, u;
        char auxH[3], auxS[3], auxV[3];

        for (i = 0; i < nbrpolys; i++)
        {
                /*if "i" is single valent AND its containment_level is odd, then...*/
                if (polyclr[i].valence == 1 && ((polyclr[i].containment_level == 1) ||
((polyclr[i].containment_level%2) != 0)))
                {
                        /*...merge it with its parent (enclosing polygon)*/
                        polyclr[i].H = polyclr[polyclr[i].parent].H;
                        polyclr[i].S = polyclr[polyclr[i].parent].S;
                        polyclr[i].V = polyclr[polyclr[i].parent].V;
                        polyclr[i].FillStyle = 0;
                } /*if*/
                /*else
                {
                        /*colour all non-single valent polygons at an odd
containment_level in grey
                        if (polyclr[i].valence != 1 && ((polyclr[i].containment_level
== 1) || ((polyclr[i].containment_level%2) != 0)))
                        {
                                polyclr[i].H = 0;
                                polyclr[i].S = 0;
                                polyclr[i].V = 70;
                        }
                }/*else*/
```

```
            /*if "i" is a bi-valent polygon at an odd containment_level*/
            if (polyclr[i].valence == 2 && ((polyclr[i].containment_level == 1) ||
((polyclr[i].containment_level%2) != 0)))
            {
                    for (t = adj[i]; t != NULL; t = t->next)
                    {
                            /*take i's non-parent adjacent polygon*/
                            if (t->v != polyclr[i].parent)
                                    /*does it belong to the same containment_level
as that of i's parent?*/
                                    if      (polyclr[t->v].containment_level      ==
polyclr[polyclr[i].parent].containment_level)
                                            /*does it belong to the same urban-set
as that of i's parent?*/
                                            if   (polyclr[t->v].containment_unit   ==
polyclr[polyclr[i].parent].containment_unit)
                                            {
                                                    /*merge "i" with its parent (and
in pratice with its non-parent adjacent polygon too)*/
                                                    polyclr[i].H                    =
polyclr[polyclr[i].parent].H;
                                                    polyclr[i].S                    =
polyclr[polyclr[i].parent].S;
                                                    polyclr[i].V                    =
polyclr[polyclr[i].parent].V;
                                                    polyclr[i].FillStyle = 0;
                                            } /*if*/
                                            /*if  i's  non-parent  adjacent  polygon
belongs to a different urban-set*/
                                            else
                                            {
                                                    /*merge "i" with its parent,...*/
                                                    polyclr[i].H                    =
polyclr[polyclr[i].parent].H;
                                                    polyclr[i].S                    =
polyclr[polyclr[i].parent].S;
                                                    polyclr[i].V                    =
polyclr[polyclr[i].parent].V;
                                                    polyclr[i].FillStyle = 0;
                                                    /*...force  the  non-parent  to  be
merged with them too, and...*/
                                                    polyclr[t->v].H                 =
polyclr[polyclr[i].parent].H;
                                                    polyclr[t->v].S                 =
polyclr[polyclr[i].parent].S;
                                                    polyclr[t->v].V                 =
polyclr[polyclr[i].parent].V;
                                                    /*...if  the  non-parent  polygon
has dead-end islands*/
                                                    for (u = adj[t->v]; u != NULL; u
= u->next)
                                                            if  (polyclr[u->v].valence
== 1)
                                                            {
                                                                    /*merge   them   as
well*/
                                                                    polyclr[u->v].H  =
polyclr[polyclr[i].parent].H;
                                                                    polyclr[u->v].S  =
polyclr[polyclr[i].parent].S;
                                                                    polyclr[u->v].V  =
polyclr[polyclr[i].parent].V;
                                                                    polyclr[u-
>v].FillStyle = 0;
                                                            } /*if*/
                                            } /*else*/
                    } /*for*/
            } /*if*/

            /* writes in the output file the final attributes of the vertex 'i':
            ** ID, colour components (H,S,V), respective colour unique value,
            ** and the colour filling style */
            if (polyclr[i].H < 10)
                    sprintf (auxH, "00%d", polyclr[i].H);
            else
```

```
                        if (polyclr[i].H < 100)
                                sprintf (auxH, "0%d", polyclr[i].H);
                        else
                                sprintf (auxH, "%d", polyclr[i].H);
                if (polyclr[i].S < 10)
                        sprintf (auxS, "00%d", polyclr[i].S);
                else
                        if (polyclr[i].S < 100)
                                sprintf (auxS, "0%d", polyclr[i].S);
                        else
                                sprintf (auxS, "%d", polyclr[i].S);
                if (polyclr[i].V < 10)
                        sprintf (auxV, "00%d", polyclr[i].V);
                else
                        if (polyclr[i].V < 100)
                                sprintf (auxV, "0%d", polyclr[i].V);
                        else
                                sprintf (auxV, "%d", polyclr[i].V);
                fprintf(ofp,  "%d  %d  %d  %d  %s%s%s  %d\n",  i  +  1,  polyclr[i].H,
polyclr[i].S, polyclr[i].V, auxH, auxS, auxV, polyclr[i].FillStyle);

        } /*for*/
} /*rationalization*/

main()
{
        int nbrpolys, nbredges, nbrtouchingedges, err, err1, *levels, i, j, aux;
        a_link  *adj, *touch;
        node_colours *polyclr;
        char *visited, *filename, *wspc, option1, option2, option3;
        FILE *ofp;

        printf("Total number of gradient regions? \n");
        do
        {
                scanf("%d", &nbrpolys);

        } while (nbrpolys < 0 || nbrpolys > 20000);

        printf("Traverse adjacencies graph using Breadth-first Search algorithm(B) or
Depth-first Search algorithm(D)? (If you don't want to obtain traversal tree press
'N' or 'n' instead)\n");
        do
        {
                scanf("%c", &option2);

        } while (option2 != 'B' && option2 != 'b' && option2 != 'D' && option2 != 'd'
&& option2 != 'N' && option2 != 'n');

        if (option2 == 'B' || option2 == 'b' || option2 == 'D' || option2 == 'd')
        {
                printf("Traversal-tree  output  file  as  a  Text  Data  File(T)  or  as  a
Commented File(C)? \n");
                do
                {
                        scanf("%c", &option1);

                } while (option1 != 'T' && option1 != 't' && option1 != 'C' && option1
!= 'c');
        }

        /* dynamic allocation of memory & creation of the graph of adjacencies, adj[i]
*/
        nbredges = 0;
        adj = (struct node **) malloc(nbrpolys*sizeof (a_link));
        err = INIT_adj(nbrpolys, &nbredges, adj);
        if (err != 0)
        {
                exit(err);
        }

        /* dynamic allocation of memory & creation of the graph of touching steep
polyogns, touch[i] */
        nbrtouchingedges = 0;
        touch = (struct node **) malloc(nbrpolys*sizeof (a_link));
        err1 = INIT_touch(nbrpolys, &nbrtouchingedges, touch);
```

```
        if (err1 != 0)
        {
                exit(err1);
        }

        /* dynamic allocation of memory & initialisation of "visited" and "levels" */
        visited = (char *) malloc(nbrpolys*sizeof (char)); /*array indicating which
vertices of adj[i] have/have not been visited*/
        levels = (int *) malloc((nbrpolys*nbrpolys)*sizeof (int));
        for (i = 0; i < nbrpolys ; i++)
                for (j = 0; j < nbrpolys; j++)
                        *(levels + i*nbrpolys + j) = -1;

        if (nbredges == nbrpolys)
        {
                printf("Number of edges equals number of vertices => graph has cycles!
Detect them? (Y/N) \n");
                do
                {
                        scanf("%c", &option3);

                } while (option3 != 'Y' && option3 != 'y' && option3 != 'N' && option3
!= 'n');
        }
        else
                if (nbredges > nbrpolys)
                {
                        printf("Number of edges is greater than number of vertices =>
graph has cycles! Detect them? (Y/N) \n");
                        do
                        {
                                scanf("%c", &option3);

                        } while (option3 != 'Y' && option3 != 'y' && option3 != 'N' &&
option3 != 'n');
                }

        /* Search the graph of adjacencies, starting from a given vertex 'i' */
        for (i = 0; i < nbrpolys; i++)
        {
                /* if the traversal-tree is supposed to be drawn then */
                if (option2 != 'N' && option2 != 'n')
                {
                        /* creates the output file to write a traversal tree in */
                        filename = create_spantree_file(i, nbrpolys, option2);
                        /* opens the output file created above */
                        if ((ofp = fopen(filename, "w")) == NULL)
                                printf("Couldn't open %s file!!!\n", filename);
                        else printf("Output file %s opened successfully!\n", filename);

                        /* allocation of memory & initialisation of the string 'wspc'
*/
                        if (option1 == 'C' || option1 == 'c')
                        {
                                aux = LEVELSPACES + 1;
                                wspc = malloc(aux*sizeof(char));
                                wspc[0] = '\0';
                        } /*if*/

                        /* all vertices are tagged as "not visited" yet -> 0 */
                        for (i = 0; i < nbrpolys; i++)
                                visited[i] = 0;

                        /* the graph is traversed from vertex 'i' */
                        if (option2 == 'D' || option2 == 'd')
                                depth_first_search(i, visited, -1, wspc, levels, i,
adj, nbrpolys, ofp, option1);
                        else breadth_first_search(i, visited, -1, wspc, levels, i, adj,
nbrpolys, ofp, option1);

                        fclose(ofp);
                        free(filename);
                } /*if*/

                /* detection of cylces in the graph adj[i] */
                /*if (option3 == 'Y' || option3 == 'y')
```

```
		{
			/* creates the output file to write the detected cycles in */
			/*filename = create_cycles_file(i, nbrpolys);
			/* opens the output file created above */
			/*if ((ofp = fopen(filename, "w")) == NULL)
				printf("Couldn't open %s file!!!\n", filename);
			else printf("Output file %s opened successfully!\n", filename);

			/* all vertices are tagged as "not visited" yet -> 0 */
			for (i = 0; i < nbrpolys; i++)
				visited[i] = 0;
			CycleDetection_DFS(i, i, visited, -1, levels, i, adj, nbrpolys,
ofp);
			fclose(ofp);
			free(filename);
		}*/

		/* creating the output file for the containment-first search tree */
		filename = create_CFStree_file(i, nbrpolys);
		/* opens the output file created above */
		if ((ofp = fopen(filename, "w")) == NULL)
			printf("Couldn't open %s file!!!\n", filename);
		else printf("Output file %s opened successfully!\n", filename);

		/* dynamic allocation of memory for the array 'polyclr' to store
vertices' attributes */
		polyclr = (struct colour *) malloc(nbrpolys*sizeof (node_colours));
		/* initialisation of this array */
		for (j = 0; j < nbrpolys; j++)
		{
			polyclr[j].containment_level = -9;
			polyclr[j].containment_unit = -9;
			polyclr[j].valence = -9;
			polyclr[j].parent = -9;
			polyclr[j].H = -9;
			polyclr[j].S = -9;
			polyclr[j].V = -9;
		}
		/* detecting different levels of containment: which polygons in the
map contain which? */
		containment_first_search(i, visited, i, adj, touch, polyclr, nbrpolys,
ofp, option2);
		fclose(ofp);
		free(filename);

		/* creating the output file for the results of the urban topology
analysis */
		filename = create_CFSRationalization_file(i, nbrpolys);
		/* opens the output file created above */
		if ((ofp = fopen(filename, "w")) == NULL)
			printf("Couldn't open %s file!!!\n", filename);
		else printf("Output file %s opened successfully!\n", filename);
		rationalization(nbrpolys, adj, polyclr, ofp);

		if (option1 == 'C' || option1 == 'c')
			free(wspc);
		fclose(ofp);
		free(filename);

		free (polyclr);
		polyclr = NULL;

	} /* for - "graph traversal" */

	printf("Task accomplished successfully!\n");
	printf("Check results...\n");
}
```

F Extract from a file of a graph spanning tree

3 304
3 303
3 301
3 300
3 299
3 297
3 295
3 291
3 290
3 286
3 278
3 276
3 272
3 270
3 269
3 266
3 265
3 262
3 261
3 255
3 251
3 249
3 248
3 246
3 244
3 241
3 237
3 236
3 235
3 233
3 231
3 230
3 229
3 228
3 226
3 224
3 223
3 221
3 218
3 216
3 213
3 212
3 210
3 209
3 208
3 205
3 203
3 202
3 201
3 198
3 196

3 195
3 191
3 189
3 185
3 182
3 178
3 171
3 169
3 167
3 165
3 161
3 156
3 155
3 154
3 153
3 150
3 149
3 148
3 147
3 146
3 145
3 141
3 140
3 138
3 137
3 135
3 133
3 132
3 131
3 130
3 129
3 127
3 126
3 125
3 124
3 122
3 121
3 120
3 118
3 117
3 116
3 113
3 112
3 110
3 109
3 106
3 100
3 99
3 98
3 94
3 90
3 88
3 84
3 81
3 78
3 77
3 76
...
[*190 lines*]
...
97 142
97 134
97 123
97 119
13 19
172 183
172 176
260 257
163 164

G Extract from a file of graph cycles

```
3 304
304 305
Pointing back to parent: 305 304
305 303
Pointing back to parent: 303 305
A cycle detected: 303 3
305 241
Pointing back to parent: 241 305
241 298
Pointing back to parent: 298 241
241 294
Pointing back to parent: 294 241
241 289
Pointing back to parent: 289 241
241 288
288 287
Pointing back to parent: 287 288
287 279
279 296
296 292
Pointing back to parent: 292 296
292 46
Pointing back to parent: 46 292
46 285
285 284
Pointing back to parent: 284 285
284 240
240 302
Pointing back to parent: 302 240
302 1
Pointing back to parent: 1 302
1 293
A cycle detected: 293 240
Pointing back to parent: 293 1
1 264
264 278
Pointing back to parent: 278 264
A cycle detected: 278 3
264 235
Pointing back to parent: 235 264
A cycle detected: 235 3
Pointing back to parent: 264 1
1 165
165 96
96 182
Pointing back to parent: 182 96
A cycle detected: 182 3
Pointing back to parent: 96 165
96 125
Pointing back to parent: 125 96
A cycle detected: 125 3
```

```
96 107
Pointing back to parent: 107 96
96 103
Pointing back to parent: 103 96
103 91
Pointing back to parent: 91 103
91 5
5 115
Pointing back to parent: 115 5
5 114
Pointing back to parent: 114 5
5 111
Pointing back to parent: 111 5
5 108
Pointing back to parent: 108 5
5 105
Pointing back to parent: 105 5
A cycle detected: 5 96
5 93
93 101
A cycle detected: 101 96
Pointing back to parent: 101 93
93 92
A cycle detected: 92 96
Pointing back to parent: 92 93
92 83
Pointing back to parent: 83 92
83 82
82 85
Pointing back to parent: 85 82
A cycle detected: 85 5
Pointing back to parent: 82 83
82 79
Pointing back to parent: 79 82
A cycle detected: 79 5
A cycle detected: 83 5
Pointing back to parent: 93 5
Pointing back to parent: 5 91
5 89
Pointing back to parent: 89 5
5 87
Pointing back to parent: 87 5
A cycle detected: 5 85
A cycle detected: 5 83
5 80
Pointing back to parent: 80 5
A cycle detected: 5 79
...
[633 lines]
...
Pointing back to parent: 39 3
3 34
Pointing back to parent: 34 3
3 32
Pointing back to parent: 32 3
3 31
Pointing back to parent: 31 3
A cycle detected: 3 30
A cycle detected: 3 15
A cycle detected: 3 14
A cycle detected: 3 12
A cycle detected: 3 11
A cycle detected: 3 6
A cycle detected: 3 5
A cycle detected: 3 4
A cycle detected: 3 2
A cycle detected: 3 1
```

H Extract from a CFS and Ratinalization output file

1 351 43 100 351043100 1
2 135 43 100 135043100 1
3 60 0 100 060000100 0
4 135 43 100 135043100 1
5 135 43 100 135043100 1
6 135 43 100 135043100 1
7 135 86 100 135086100 0
8 135 86 100 135086100 0
9 135 86 100 135086100 0
10 135 86 100 135086100 0
11 135 43 100 135043100 1
12 135 43 100 135043100 1
13 351 86 100 351086100 0
14 189 32 100 189032100 1
15 189 32 100 189032100 1
16 135 86 100 135086100 0
17 135 86 100 135086100 0
18 135 86 100 135086100 0
19 351 86 100 351086100 0
20 135 86 100 135086100 0
21 135 86 100 135086100 0
22 189 64 100 189064100 0
23 135 86 100 135086100 0
24 135 86 100 135086100 0
25 135 86 100 135086100 0
26 189 64 100 189064100 0
27 135 86 100 135086100 0
28 135 86 100 135086100 0
29 135 86 100 135086100 0
30 135 43 100 135043100 1
31 60 0 100 060000100 0
32 60 0 100 060000100 0
33 135 86 100 135086100 0
34 60 0 100 060000100 0
35 189 64 100 189064100 0
36 135 86 100 135086100 0
37 135 86 100 135086100 0
38 135 100 71 135100071 1
39 60 0 100 060000100 0
40 135 43 100 135043100 1
41 135 86 100 135086100 0
42 60 0 100 060000100 0
43 135 86 100 135086100 0
44 135 86 100 135086100 0
45 60 0 100 060000100 0
46 27 43 100 027043100 1
47 135 86 100 135086100 0
48 135 86 100 135086100 0
49 135 86 100 135086100 0
50 135 86 100 135086100 0
51 60 0 100 060000100 0

```
52 60 0 100 060000100 0
53 135 86 100 135086100 0
54 135 86 100 135086100 0
55 135 86 100 135086100 0
56 60 0 100 060000100 0
57 60 0 100 060000100 0
58 135 43 100 135043100 1
59 135 86 100 135086100 0
60 135 86 100 135086100 0
61 135 43 100 135043100 1
62 135 86 100 135086100 0
63 60 0 100 060000100 0
64 135 86 100 135086100 0
65 135 86 100 135086100 0
66 135 100 71 135100071 1
67 60 0 100 060000100 0
68 60 0 100 060000100 0
69 135 86 100 135086100 0
70 60 0 100 060000100 0
71 135 86 100 135086100 0
72 135 86 100 135086100 0
73 189 64 100 189064100 0
74 324 65 100 324065100 1
75 324 100 70 324100070 0
76 324 65 100 324065100 1
77 60 0 100 060000100 0
78 60 0 100 060000100 0
79 135 86 100 135086100 0
80 135 86 100 135086100 0
81 60 0 100 060000100 0
82 135 100 71 135100071 1
83 135 86 100 135086100 0
84 60 0 100 060000100 0
85 135 86 100 135086100 0
86 189 64 100 189064100 0
87 135 86 100 135086100 0
88 60 0 100 060000100 0
89 135 86 100 135086100 0
90 60 0 100 060000100 0
91 135 86 100 135086100 0
92 135 100 71 135100071 1
93 135 86 100 135086100 0
94 60 0 100 060000100 0
95 189 64 100 189064100 0
96 135 86 100 135086100 0
97 351 86 100 351086100 0
98 60 0 100 060000100 0
99 60 0 100 060000100 0
100 60 0 100 060000100 0
101 135 86 100 135086100 0
102 189 64 100 189064100 0
103 135 86 100 135086100 0
104 189 64 100 189064100 0
105 135 86 100 135086100 0
106 60 0 100 060000100 0
107 135 86 100 135086100 0
108 135 86 100 135086100 0
109 60 0 100 060000100 0
110 60 0 100 060000100 0
111 135 86 100 135086100 0
...
[182 lines]
...
300 60 0 100 060000100 0
301 60 0 100 060000100 0
302 351 86 100 351086100 0
303 27 43 100 027043100 1
304 27 43 100 027043100 1
305 27 86 100 027086100 0
```

I Visualisation of the spatial topology analyses: the interactive tool code

Module1

```
Sub StartEvents()

Message = MsgBox("Select new feature(s) on the map...", vbOKOnly, "Starting Urban
Topology Analysis")
UserForm1.StartEvents

End Sub
```

UserForm1

```
Dim WithEvents g_Map As esriCore.map

---------------------------------------------------------------------
Public Sub StartEvents()

Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Set g_Map = pMxDoc.FocusMap

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub SearchButton_Click()
' initialise tree view
' then show spanning tree

If VerticesList.ListIndex <> -1 Then 'only continue if 1 polygon selected
    UserForm2.Show 0
Else
    Message = MsgBox("Select from the list the polygon to be set the spanning tree's
root", vbOKOnly, "Graph traversal: generating a spanning tree...")
End If
End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub ClearButton_Click()

UserForm1.VerticesList.Clear
UserForm1.VerticesList.AddItem "No list"
```

```
End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub InfoButton_Click()

If VerticesList.ListIndex <> -1 Then 'only continue if 1 polygon selected
    UserForm3.Show 0
Else
    Message = MsgBox("Select from the list the polygon to be identified...",
vbOKOnly, "Polygon information")
End If

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub Frame1_Click()

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub g_Map_SelectionChanged()

If Not UserForm1.Visible Then
    UserForm1.Show vbModeless
    UserForm1.Caption = "Urban Topology Analysis"
End If

'clear any previous results in the list
UserForm1.VerticesList.Clear

'get the map's selection
Dim activeView As IActiveView
Set activeView = g_Map

'enumerate over the selected features
Dim featureEnum As IEnumFeature
Set featureEnum = activeView.Selection

featureEnum.Reset 'reset the cursor

'in order to access other field values besides the shape field
Dim pEnumFeatureSetup As IEnumFeatureSetup
Set pEnumFeatureSetup = featureEnum
pEnumFeatureSetup.AllFields = True

Dim index As Long
Dim feat As IFeature
Set feat = featureEnum.Next
index = feat.Fields.FindField("ARCS45COPY#")
  Do While (Not feat Is Nothing)
    'Debug.Print feat.OID 'print unto the debugger window
    'MsgBox Str(feat.Value(index))
    UserForm1.VerticesList.AddItem Str(feat.Value(index))
    Set feat = featureEnum.Next
  Loop

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub UserForm_Click()

    StartEvents
    UserForm1.VerticesList.Clear
    UserForm1.VerticesList.AddItem "No polygons selected!"

End Sub
---------------------------------------------------------------------
```

UserForm2

```
---------------------------------------------------------------------
Private Sub CFSButton_Click()
'Given the spanning tree root, this routine colours map polygons
'according to the respective results of the urban topology analysis

Dim pDoc As IMxDocument
Set pDoc = ThisDocument
Dim pMap As IMap
Set pMap = pDoc.FocusMap

Dim pLayer As ILayer
Set pLayer = FindLayer(pMap, "arcs45copy polygon")

Dim pFLayer As IFeatureLayer
Set pFLayer = pLayer
Dim pLyr As IGeoFeatureLayer
Set pLyr = pFLayer

Dim filename_head As String
Dim filename_tail As String
Dim filename_num As String
Dim temp_filename_num As String
Dim fso As New FileSystemObject
Dim f As File
Dim fsoStream As TextStream
Dim strLine As String
Dim FirstSpace As Long
Dim i_space_position As Integer
Dim Aux1, Aux2, Aux3, Aux4 As String

'check ID# of selected polygon and continue
If UserForm1.VerticesList.Value < 10 Then
    temp_filename_num = Right(UserForm1.VerticesList.Value, 1)
    filename_num = "0" & "0" & temp_filename_num
Else
    If UserForm1.VerticesList.Value < 100 Then
        temp_filename_num = Right(UserForm1.VerticesList.Value, 2)
        filename_num = "0" & temp_filename_num
    Else
        temp_filename_num = Right(UserForm1.VerticesList.Value, 3)
        filename_num = temp_filename_num
    End If
End If

filename_head                                                                    =
"C:\ZP\PhD\Implementations\Experiments\AlgorithmsinC\GraphAnalysis\SceneAnalysis\tree
files\LiDARdata\NE_Zone\45DegreeSlope\GraphRationalization\rationalizationFrom_"
filename_tail = ".dat"

Set f = fso.GetFile(filename_head & filename_num & filename_tail)
Set fsoStream = f.OpenAsTextStream(ForReading)

'read in polygon attributes from the input file, and populate fields
'COLOUR, HUE, SATURATION, VALUE & FILLSTYLE in the Attribute Table
Do While Not fsoStream.AtEndOfStream
    'read each file line...
    strLine = fsoStream.ReadLine
    'retrieves polygon ID...
    i_space_position = InStr(strLine, " ")
    Dim polyID As String
    polyID = Left(strLine, i_space_position - 1)
    'retrieves the respective "colour unique value", "hue", "saturation", "value",
"fill style" values
    Dim Hue, Saturation, Value, UnqValueColour, FillStyle As Long
    FirstSpace = InStr(strLine, " ")
    Aux1 = Mid(strLine, FirstSpace + 1)
    FirstSpace = InStr(Aux1, " ")
    Hue = CLng(Left(Aux1, FirstSpace - 1))
    Aux2 = Mid(Aux1, FirstSpace + 1)
    FirstSpace = InStr(Aux2, " ")
    Saturation = CLng(Left(Aux2, FirstSpace - 1))
    Aux3 = Mid(Aux2, FirstSpace + 1)
```

```
    FirstSpace = InStr(Aux3, " ")
    Value = CLng(Left(Aux3, FirstSpace - 1))
    Aux4 = Mid(Aux3, FirstSpace + 1)
    FirstSpace = InStr(Aux4, " ")
    UnqValueColour = CLng(Left(Aux4, FirstSpace - 1))
    Debug.Print UnqValueColour
    i_space_position = InStrRev(strLine, " ")
    FillStyle = Right(strLine, Len(strLine) - i_space_position)
    Debug.Print FillStyle

    Dim pScratchWorkspace As IWorkspace
    Dim pScratchWorkspaceFactory As IScratchWorkspaceFactory
    Set pScratchWorkspaceFactory = New ScratchWorkspaceFactory
    Set pScratchWorkspace = pScratchWorkspaceFactory.DefaultScratchWorkspace

    Dim sConnectionFile As String
    sConnectionFile = "C:\ZP\PhD\Implementations\BoundsSelections\NE_Zone"

    'it will deal only with ArcInfo coverages
    Dim pFact As IWorkspaceFactory
    Set pFact = New ArcInfoWorkspaceFactory

    Dim pApp As IApplication
    Set pApp = Application

    Dim pWorkspace As IWorkspace
    Set pWorkspace = pFact.OpenFromFile(sConnectionFile, pApp.hWnd)

    Dim pFeatureWorkspace As IFeatureWorkspace
    Set pFeatureWorkspace = pWorkspace

    'open the feature class: polygon
    Dim pFeatureClass As IFeatureClass
    Set pFeatureClass = pFeatureWorkspace.OpenFeatureClass("arcs45copy:polygon")

    'select the record where polygon ID = polyID
    Dim pTable As ITable
    Set pTable = pLyr.FeatureClass

    Dim pFilter As IQueryFilter
    Set pFilter = New QueryFilter

    pFilter.SubFields = "FID, COLOUR, HUE, SATURATION, VALUE, FILLSTYLE"
    pFilter.WhereClause = "FID = " & polyID

    Dim pSelectionSet As ISelectionSet
    Set    pSelectionSet    =    pTable.Select(pFilter,    esriSelectionTypeHybrid,
esriSelectionOptionNormal, pScratchWorkspace)

    'create a cursor on the previous selection to access the selected record
    Dim pFCursor As IFeatureCursor
    pSelectionSet.Search pFilter, False, pFCursor
    'retrieve the feature
    Dim pFeature As IFeature
    Set pFeature = pFCursor.NextFeature
    Dim pFieldColour, pFieldHue, pFieldSat, pFieldVal, pFieldFS As Long
    pFieldColour = pFCursor.FindField("COLOUR")
    pFieldHue = pFCursor.FindField("HUE")
    pFieldSat = pFCursor.FindField("SATURATION")
    pFieldVal = pFCursor.FindField("VALUE")
    pFieldFS = pFCursor.FindField("FILLSTYLE")

    'open the editing session;
    'update COLOUR, HUE, SATURATION, VALUE & FILLSTYLE fields of
    'the selected record(s) and stores the updated record;
    'close the editing session.
    While Not pFeature Is Nothing
        Dim pWorkspaceEdit As IWorkspaceEdit
        Set pWorkspaceEdit = pWorkspace

        pWorkspaceEdit.StartEditing True
        Dim pRow As esriCore.IRow
        Set pRow = pTable.GetRow(pFeature.OID)
        pRow.Value(pFieldColour) = UnqValueColour
        pRow.Value(pFieldHue) = Hue
        pRow.Value(pFieldSat) = Saturation
```

```
        pRow.Value(pFieldVal) = Value
        pRow.Value(pFieldFS) = FillStyle
        pRow.Store
        pWorkspaceEdit.StartEditing False

        Set pFeature = pFCursor.NextFeature
    Wend
Loop 'Do While Not

'now that the field "COLOUR" is populated with unique colour class values
'make the renderer with those values & classify objects
Dim pFeatCls As IFeatureClass
Set pFeatCls = pFLayer.FeatureClass
Dim pQueryFilter As IQueryFilter
Set pQueryFilter = New QueryFilter 'empty supports SELECT * (and all features are
selected)
Dim pFeatCursor As IFeatureCursor
Set pFeatCursor = pFeatCls.Search(pQueryFilter, False)
Dim feature, n, j As Long
feature = 0
Dim ValFound As Boolean

Dim myRender As IUniqueValueRenderer
Set myRender = New UniqueValueRenderer

Dim mySymb As ISimpleFillSymbol
Set mySymb = New SimpleFillSymbol
Dim pOutLine As IRgbColor
Set pOutLine = New RgbColor
pOutLine.RGB = RGB(160, 160, 160) 'grey
mySymb.Outline.Color = pOutLine
mySymb.Outline.Width = 0.4
mySymb.Style = esriSFSSolid

'properties that should be set prior to adding values
myRender.FieldCount = 1
myRender.Field(0) = "COLOUR"
myRender.DefaultSymbol = mySymb
myRender.UseDefaultSymbol = True

'loop through all the features selected:
n = pFeatCls.FeatureCount(pQueryFilter)
Dim pFeat As IFeature
Do Until feature = n
    Set pFeat = pFeatCursor.NextFeature

    Dim myFillSym As ISimpleFillSymbol
    Set myFillSym = New SimpleFillSymbol
    myFillSym.Outline.Color = pOutLine
    myFillSym.Outline.Width = 0.4
    If pFeat.Value(pFieldFS) = 0 Then
        myFillSym.Style = esriSFSSolid 'solid colour style for flat polygons
    Else
        myFillSym.Style = esriSFSForwardDiagonal '"forward diagonal" style for steep
polygons
    End If

    Dim x As String
    x = pFeat.Value(pFieldColour)

    'check whether this value has already been added to the render;
    'if not then add it
    ValFound = False
    For j = 0 To (myRender.ValueCount - 1)
        If myRender.Value(j) = x Then
            ValFound = True
            Exit For
        End If
    Next j
    'if the current UniqueValue is new then
    'create the respective symbol and add it to the UniqueValue list
    If Not ValFound Then
        'creating a new HSVcolour object
        Dim myHSVcolour As IHsvColor
        Set myHSVcolour = New HsvColor
```

```
        myHSVcolour.Hue = pFeat.Value(pFieldHue)
        myHSVcolour.Saturation = pFeat.Value(pFieldSat)
        myHSVcolour.Value = pFeat.Value(pFieldVal)

        myFillSym.Color = myHSVcolour

        myRender.AddValue x, "", myFillSym
        myRender.Label(x) = x
        myRender.Symbol(x) = myFillSym
    End If
    feature = feature + 1
Loop 'Do Until

'With the created list of colour UniqueValues & associated symbols,
'redisplay map features accordingly
myRender.ColorScheme = "Costum"
myRender.FieldType(0) = True
Set pLyr.Renderer = myRender
pLyr.DisplayField = "COLOUR"
'make the layer properties symbology
'to show the correct interface
Dim pRPP As IRendererPropertyPage
Set pRPP = New UniqueValuePropertyPage
pLyr.RendererPropertyPageClassID = pRPP.ClassID
'refresh the TOC
pDoc.activeView.ContentsChanged
pDoc.UpdateContents
'draw the map
pDoc.activeView.Refresh

End Sub
--------------------------------------------------------------------

--------------------------------------------------------------------
Private Sub FindButton_Click()

Dim index As Integer
For index = 1 To nbrpolys
    SpanningTree.Nodes.Item("N" & index).Bold = False
Next index
SpanningTree.Refresh

End Sub
--------------------------------------------------------------------

--------------------------------------------------------------------
Private Sub FindPoly_Change()

End Sub
--------------------------------------------------------------------

--------------------------------------------------------------------
Private Sub SpanningTree_NodeClick(ByVal Node As MSComctlLib.Node)

SpanningTree.SelectedItem.Expanded = True
SpanningTree.SelectedItem.Selected = True

Dim pMxApp As IMxApplication
Dim pMxDoc As IMxDocument
Set pMxDoc = Application.Document
Dim pMap As IMap
Set pMap = pMxDoc.FocusMap

'get the selected Tree node's ID# and continue
Dim polyID As String
polyID = SpanningTree.SelectedItem.Text

'find the layer & attribute table we're going to select from
Dim lyr As IFeatureLayer
Set lyr = FindLayer(pMap, "arcs45copy polygon")
'Set lyr = pMap.Layer(0)
Dim pTable As ITable
Set pTable = lyr.FeatureClass

'create a query filter
Dim pFilter As IQueryFilter
```

```
Set pFilter = New QueryFilter
'set up the "where" clause
pFilter.SubFields = "FID"
pFilter.WhereClause = "FID = " & polyID

'highlighting a polygon on the map
Dim pFSel As IFeatureSelection
Set pFSel = lyr
pFSel.SelectFeatures pFilter, esriSelectionResultNew, True
pMxDoc.activeView.Refresh

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub TextBox1_Change()

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub UserForm_Activate()
'given a specific root, this routine pics up the respective tree edges file and
'initialises tree view;
'shows spanning tree.

Dim fso As New FileSystemObject
Dim f As File
Dim fsoStream As TextStream
Dim strLine As String
Dim i_space_position As Integer
Dim left_node As Integer
Dim right_node As Integer
Dim filename_head As String
Dim filename_tail As String
Dim filename_num As String
Dim temp_filename_num As String
Dim nodx As Node
Dim nbrpolys As Integer

'check ID# of selected polygon and continue
If UserForm1.VerticesList.Value < 10 Then
    temp_filename_num = Right(UserForm1.VerticesList.Value, 1)
    filename_num = "0" & "0" & temp_filename_num
Else
    If UserForm1.VerticesList.Value < 100 Then
        temp_filename_num = Right(UserForm1.VerticesList.Value, 2)
        filename_num = "0" & temp_filename_num
    Else
        temp_filename_num = Right(UserForm1.VerticesList.Value, 3)
        filename_num = temp_filename_num
    End If
End If

'read input from a text file

filename_head                                                                    =
"C:\ZP\PhD\Implementations\Experiments\AlgorithmsinC\GraphAnalysis\SceneAnalysis\tree
files\LiDARdata\NE_Zone\45DegreeSlope\breadth_first_search\outputtree_"
filename_tail = ".dat"

Set f = fso.GetFile(filename_head & filename_num & filename_tail)
Set fsoStream = f.OpenAsTextStream(ForReading)

'add tree's root/first node
SpanningTree.Nodes.Clear
Set nodx = SpanningTree.Nodes.Add(, , "N" & temp_filename_num, temp_filename_num)
nodx.Expanded = True 'expand root branch
SpanningTree.Nodes.Item("N" & temp_filename_num).Selected = True 'select root node

nbrpolys = 0 'starts counting number of polygons

'Read the file line by line, printing the results to the debug
'to extract text, use left, right and instr commands
Do While Not fsoStream.AtEndOfStream
    strLine = fsoStream.ReadLine
```

```
    i_space_position = InStr(strLine, " ")
    left_node_id = Left(strLine, i_space_position - 1)
    right_node_id = Right(strLine, Len(strLine) - i_space_position)
    If left_node_id > nbrpolys Then
        nbrpolys = left_node_id
    End If
    If right_node_id > nbrpolys Then
        nbrpolys = right_node_id
    End If
    Set nodx = SpanningTree.Nodes.Add("N" & left_node_id, tvwChild, "N" &
right_node_id, right_node_id)
    nodx.Expanded = True
Loop

SpanningTree.BorderStyle = ccFixedSingle
SpanningTree.Indentation = 20

End Sub
---------------------------------------------------------------------

---------------------------------------------------------------------
'function to find a particular layer in a given map
Function FindLayer(map As IMap, nome As String) As ILayer

    Dim i As Integer
    For i = 0 To map.LayerCount - 1
        If map.Layer(i).name = nome Then
            Set FindLayer = map.Layer(i)
            Exit Function
        End If
    Next i

End Function
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub SpanningTree_Click()

End Sub
---------------------------------------------------------------------
```

UserForm3

```
---------------------------------------------------------------------
'function to find a particular layer in a given map
Function FindLayer(map As IMap, nome As String) As ILayer

    Dim i As Integer
    For i = 0 To map.LayerCount - 1
        If map.Layer(i).name = nome Then
            Set FindLayer = map.Layer(i)
            Exit Function
        End If
    Next i

End Function
---------------------------------------------------------------------

---------------------------------------------------------------------
Private Sub UserForm_Initialize()

Dim pMxDoc As IMxDocument
Set pMxDoc = Application.Document
Dim pMap As IMap
Set pMap = pMxDoc.FocusMap
Dim pActiveView As IActiveView
Set pActiveView = pMxDoc.FocusMap
Dim pScreenDisplay As IScreenDisplay
Set pScreenDisplay = pActiveView.ScreenDisplay

'get the ID# from the VerticesList ListBox and continue
Dim polyID As String
If UserForm1.VerticesList.Value < 10 Then
```

```
    polyID = Right(UserForm1.VerticesList.Value, 1)
Else
    If UserForm1.VerticesList.Value < 100 Then
        polyID = Right(UserForm1.VerticesList.Value, 2)
    Else
        polyID = Right(UserForm1.VerticesList.Value, 3)
    End If
End If

'find the layer & attribute table we're going to select from
Dim lyr As IFeatureLayer
Set lyr = FindLayer(pMap, "arcs45copy polygon") 'defined myself this function
Dim pTable As ITable
Set pTable = lyr.FeatureClass

'create a query filter
Dim pFilter As IQueryFilter
Set pFilter = New QueryFilter
'set up the "where" clause
pFilter.SubFields = "FID, AREA, PERIMETER, SLOPECLASS"
pFilter.WhereClause = "FID = " & polyID

'select respective record & access its attributes
Dim pSelectionSet As ISelectionSet
Set     pSelectionSet     =     pTable.Select(pFilter,     esriSelectionTypeHybrid,
esriSelectionOptionNormal, Nothing)
Dim pFCursor As IFeatureCursor
Set pFCursor = pTable.Search(pFilter, False)

Dim pFeat As IFeature
Set pFeat = pFCursor.NextFeature

Dim pFieldIndex1, pFieldIndex2, pFieldIndex3, pFieldIndex4 As Long
pFieldIndex1 = pFCursor.FindField("FID") 'The table's field being searched
pFieldIndex2 = pFCursor.FindField("PERIMETER") 'The table's field being searched
pFieldIndex3 = pFCursor.FindField("AREA") 'The table's field being searched
pFieldIndex4 = pFCursor.FindField("SLOPECLASS") 'The table's field being searched

'highlighting polygon feature
Dim pFSel As IFeatureSelection
Set pFSel = lyr
pFSel.SelectFeatures pFilter, esriSelectionResultNew, True
pMxDoc.activeView.Refresh

While Not pFeat Is Nothing
    Label5.Caption = Str(pFeat.Value(pFieldIndex1))
    Label6.Caption = Str(pFeat.Value(pFieldIndex2))
    Label7.Caption = Str(pFeat.Value(pFieldIndex3))
    If pFeat.Value(pFieldIndex4) = 1 Then
        Label8.Caption = " Steep polygon"
    Else
        Label8.Caption = " Flat polygon"
        'Label8.Caption = Str(pfeat.Value(pFieldIndex4))
        'MsgBox "ID = " & pfeat.Value(pFieldIndex1) & vbNewLine & "PERIMETER = " &
    pfeat.Value(pFieldIndex2) & vbNewLine & "AREA = " & pfeat.Value(pFieldIndex3) &
    vbNewLine & "SLOPE CLASS = " & pfeat.Value(pFieldIndex4)
    End If
    Set pFeat = pFCursor.NextFeature
Wend

End Sub
```

J Paper in press for publication

This appendix includes a paper accepted for publication (2007) in a special issue on Topology and Topological Data Structures of the *Computers, Environment and Urban Systems* journal, Elsevier Science Ltd.

Graph theory in higher order topological analysis of urban scenes

de Almeida[a b], J-P; Morley[a], J G; Dowman, I J[a]

[a] Dept of Geomatic Engineering, University College London, Gower Street, LONDON WC1E 6BT, UK (almeida, jmorley, idowman)@ge.ucl.ac.uk
Tel. +44 (0)20 7679 2740 Fax +44 (0)20 7380 0453
http://www.ge.ucl.ac.uk

[b] Geomatic Engineering Group, Dept of Mathematics, Faculty of Science and Technology, University of Coimbra, Largo D. Dinis, Apartado 3008, 3001-454 COIMBRA, Portugal
Tel. +351 239 791 150 Fax +351 239 832 568
http://www.mat.uc.pt

Abstract

Interpretation and analysis of spatial phenomena is a highly time consuming and laborious task in several fields of the Geomatics world. That is why the automation of these tasks is especially needed in areas such as GISc. Carrying out those tasks in the context of an urban scene is particularly challenging given the complex spatial pattern of its elements. The aim of retrieving structured information from an initial unstructured data set translated into more meaningful homogeneous regions can be achieved by identifying meaningful structures within the initial collection of objects, and by understanding their topological relationships and spatial arrangement. This task is being accomplished by applying graph theory and by performing urban scene topology analysis. For this purpose a graph-based system is being developed, and LiDAR data are currently being used as an example scenario. A particular emphasis is being given to the visualisation aspects of graph analysis, as visual inspections can often reveal patterns not discernable by current automated analysis techniques. This paper focuses primarily on the role of graph theory in the design of such a tool for the analysis of urban scene topology.

Keywords: GIS, Topology, Graph theory, Analysis, Visualisation, Understanding

1. Introduction

Interpretation and analysis of spatial phenomena is a highly time consuming and laborious task in several fields of the Geomatics world. This is particularly evident given the more accurate, but also the increasingly large, spatial data sets that are being acquired with the new technologies continuously being developed, *e.g.* LiDAR data, (Anders *et al.*, 1999). In addition, these tasks become extremely complex when the starting point is an unstructured data set. That is why the automation of these tasks is especially needed in areas such as Geographical Information Science (GISc).

According to some authors (including Eyton, 1993, and Barr & Barnsley, 1996, both cited in Barnsley & Barr, 1998), "the classification process of spatial information to produce land-cover maps for urban areas can be considered fairly straightforward if we compare it with the process of deriving information from those maps on urban land-use, which is normally much more problematic". Carrying out this sort of

analysis in the context of an urban scene is particularly challenging given the great number of component elements (*e.g.* buildings, roads and intra-urban open spaces) and their generally complex spatial pattern.

The aim of retrieving structured information translated into more meaningful homogeneous regions, for instance from an initial unstructured data set, can be achieved by identifying meaningful structures within the initial random collection of objects and by understanding their spatial arrangement (Anders *et al.*, 1999). It is believed that the task of understanding topological relationships between objects can be accomplished by applying graph theory and carrying out graph analysis.

A graph-based system for urban scene analysis is still being developed, and this paper describes primarily the role of graph theory in the design of such a tool. The paper is structured as follows. After giving a brief overview of the context of this research in sections 1.1 and 1.2, the theoretical background of our concepts is presented in section 2: the individual steps for the preparation of the unstructured data are identified in section 2.1; details of the construction of the network of connectivity are given in section 2.2, *e.g.* retrieval of polygon adjacency information and how adjacency graphs are represented in the computer; finally, the bases for the analytical analysis method are described in section 2.3. To conclude the paper, an outlook on the next steps of our work is given (especially aspects of the visualisation of the urban scene topology and its analysis are presented).

1.1 Topology

Topology is a particularly important research area in the field of GISc, for it is a central defining feature of a geographical information system (GIS) (Reed, 1999; Theobald, 2001). But, as far as topological relationships between spatial objects are concerned, generally speaking contemporary desktop GIS packages do not support further information beyond the first level of adjacency (Theobald, 2001).

Therefore, one of the first motivations of the research work described in this paper was to focus on scene analysis by building up a technique for the better understanding of topological relationships beyond the first level of adjacency, between GIS vector-based objects.

1.2 Graph theory

Another initial interest of this research was to investigate further the possible use of graph theory for the purposes mentioned in the previous section. Concepts from the mathematical areas of topology and graph theory are valuable for revealing the spatial structure of geographical entities and their spatial arrangement. In fact, these mathematical frameworks have been used so far in different applications of a wide range of fields for that purpose (Barr & Barnsley, 1997; Barnsley & Barr, 1998; Kim & Muller 1999; Bunn *et al.*, 2000; Roberts *et al.*, 2000; Bauer & Steinnocher, 2001; O'Sullivan & Turner, 2001; Nardinocchi *et al.*, 2003; Steel *et al.*, 2003; Barr & Barnsley, 2004).

Graph theory is said to be fairly powerful and elegant based only on a few basic simple principles (Temperley, 1981). Laurini and Thompson (1992) have maintained that this particular tool is "extremely valuable and efficient in storing and describing the spatial structure of geographical entities and their spatial arrangement". Theobald

(2001) added that "concepts of graph theory allow us to extend the standard notion of adjacency".

For topological analysis purposes, some geographical entities can be represented by vertices in a graph, and the connections between them by edges in a graph. The combination of vertices and edges forms a graph (Temperley, 1981; Gibbons 1989; Wilson, 1996; Gross & Yellen, 1999). In such a topological graph-based representation of a geographic dataset, information referring, for instance to line shape, compass orientation or line length, is normally thrown away concentrating on the structural components: junctions and connections (Laurini and Thompson, 1992).

2. The graph-based analysis tool

In most applications developed so far, the starting point is to some extent a meaningful data set in terms of the scene. We seek to explore and investigate whether it is possible to start at a level further back, before meaningful data sets are obtained, and hence in this case no prior knowledge of the spatial entities is being assumed.

2.1 Preparation of the polygon data base

To start with, LiDAR data are being used as an example scenario to test the graph-based technique. It is an unstructured data set with no patterns pre-defined and meaningless in terms of urban scene. The data set currently being used has 3m point spacing and contains both ground points and object points reflected from trees, buildings and other small objects above ground level. The data set refers to an area ($1470 \times 1530 m^2$) in Kew, southwest London, including the National Archives building and its neighbourhood, comprising a total of 169819 laser points. To give an idea of the spatial distribution of the range data in vertical terms, the cloud of points is colour coded in Figure 1 according to the points' height.

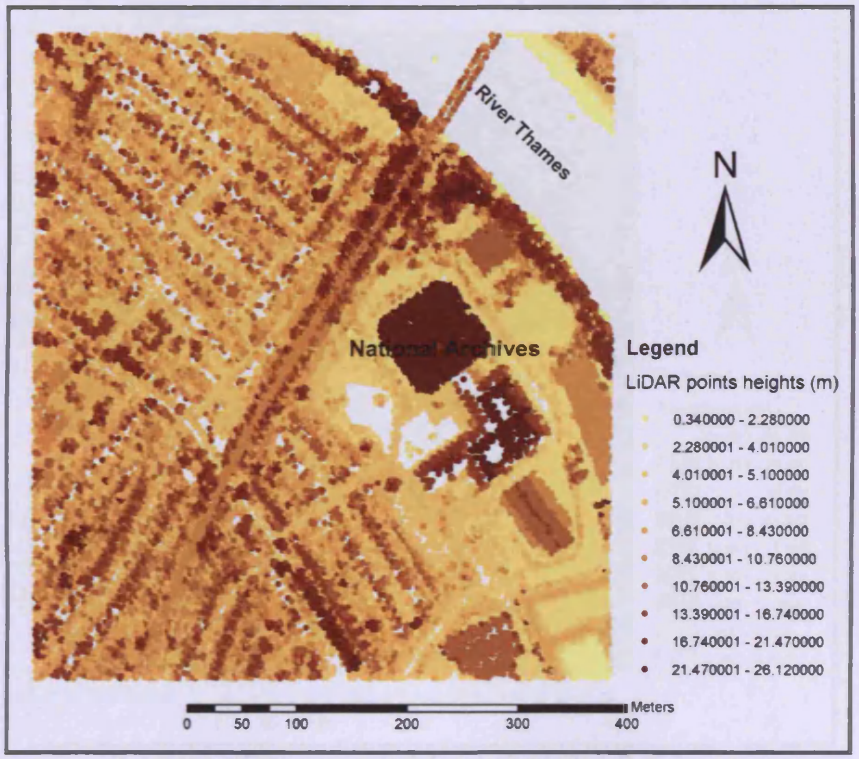


Figure 1. LiDAR data set being used - Kew, southwest London. Data is colour coded in elevation range. (After de Almeida *et al.*, 2004a, 2004b)

In order to start structuring information and make it more explicit, some topological information was brought in by establishing a triangulated irregular network (TIN) through the given data set (*vd.* Figure 2), (Toussaint, 1980a; Urquhart, 1982; Edelsbrunner *et al.*, 1983; Kirkpatrick and Radke, 1985). In fact, the generation of the TIN was based upon the Delaunay triangulation which, given the fact that it is a maximal planar description of the given point set internal structure (Kirkpatrick and Radke, 1985), expresses proximities and neighbourhoods between the LiDAR points. This was accomplished with the 3D Analyst extension of ArcMap (ArcGIS 8.3 environment), using an ArcInfo coverage containing the range point set described above as the input.

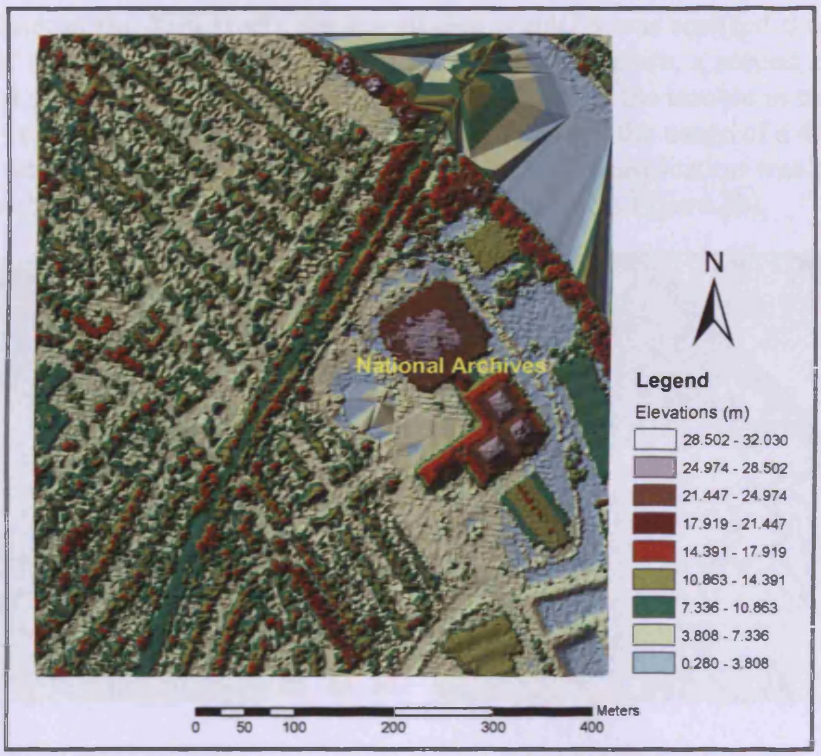


Figure 2. TIN generated from the LiDAR point set.
(After de Almeida *et al.*, 2004a, 2004b)

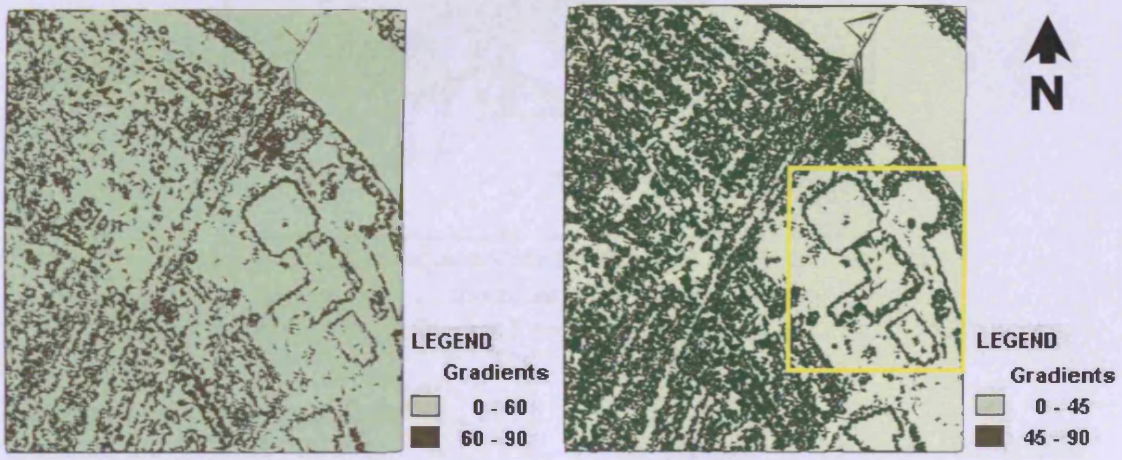
In terms of GIS analysis, a TIN translates original unstructured point data into what can be defined as a set of "first order connections" in vector domain, *i.e.* spatial relationships between nearest neighbours. By using a graph-based approach the initial aim is to build up networks of connectivity through these data sets, and hence to perform higher-order connectivity analysis. In other words, we seek to investigate and understand the spatial relationships between objects within the context of the whole scene rather than within the context of their own neighbourhood.

After the generation of the TIN a classification was applied to its facets based on their attributes. As the point spacing of this data set is about 3m on the plane, and supposing that the average height of an urban feature is about 5m, a TIN facet against an urban feature and the local terrain has roughly a 60° gradient.

The 60° gradient value was considered for a binary thresholding of the facet gradients; more classes could have been considered, but the problem is clearly simplified by considering only two gradient classes. In considering two classes of gradients, *i.e.* "flat" and "steep" TIN facets, it is expected that the most important urban features (*e.g.* man-made structures and vegetation) are enclosed by the steep facets.

Several polygonal regions were then generated by aggregating TIN facets in accordance to the binary classification mentioned above, *i.e.* facets of the same class meeting on edges were merged; facets of the same class meeting at a node were preserved. As it can be seen in Figure 3a), building features are not well defined and this is more evident in the eastern area containing the National Archives building and surrounding buildings. This fact was probably caused by the variation in facet gradient given the non uniform distribution of the LiDAR points. Moreover, after

having a look at the TIN facets slope statistics graph, it was realized that the great majority of the facets have a gradient less than 30°. Therefore, a second experiment was carried out using a lower gradient threshold; however the trouble in using such a low value is that more noise is brought in. Given this fact, the usage of a 45° gradient threshold was considered and an equal interval binary classification was performed. The result obtained with this new classification is shown in Figure 3b).



a) using the 60° gradient threshold

b) using the 45° gradient threshold

Figure 3. Binary classification of TIN facets.
(Box in yellow shows area extracted for analysis in Figure 4)

Given the large size of the initial data set and the complexity of the map of polygonal regions displayed in Figure 3b), a case study area was chosen (*vd.* yellow box in Figure 3b)) comprising the National Archives building and its neighbourhood, where urban features, like buildings and some trees, are clearly standing on their own and hence easier to analyze.

2.2 Construction of the network of connectivity

The next step was the establishment of a network of connectivity throughout this map of polygonal regions by using graph theory: each merged polygon is represented by one vertex in the graph, and graph edges link graph vertices corresponding to adjacent polygons. Figure 4 shows the corresponding graph of adjacencies for the case study area.

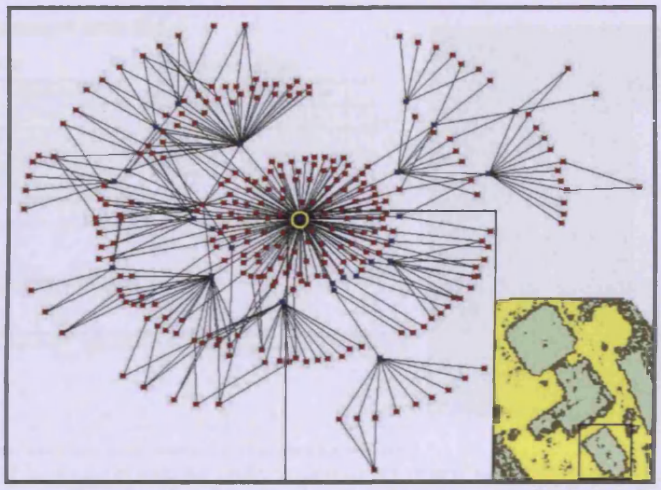


Figure 4. Graph of adjacencies for the case study area shown.
Boxes refer to the areas enlarged in Figure 6.
(Graph generated using Ucinet 6 for Windows; Borgatti *et al.*, 2002)

2.2.1 Polygon adjacencies retrieval

It should be pointed out at this stage that, for the purpose of this work, the spatial relationship of *adjacency* between polygons means that two polygons are adjacent if and only if they share at least one arc. As far as the spatial relationship of *containment* is concerned, our understanding is broader than the usual notion of containment between polygons. It is clear that a polygon is contained by another polygon if the former is completely surrounded and enclosed by the latter. However, in our case it is also possible to have a polygon surrounded by a ring of two or more polygons of the same class, which were not merged into one single entity because they happen to meet only at nodes. In this case, the former polygon is said to be *contained* by this ring of polygons.

A routine was implemented in *Arc Macro Language* (AML) to access polygon and arc attribute tables of the polygon coverage, and hence to retrieve polygon adjacencies. Given the GIS environment that is being used, this task implied a combination of information spread over two lists (*vd.* Figure 5): Polygon Component Arcs list (information referring to area definition) and the Arc Adjacent Polygons list (information referring to *connectivity* of arcs and *contiguity* of polygons), (ESRI, 1995, 2004).

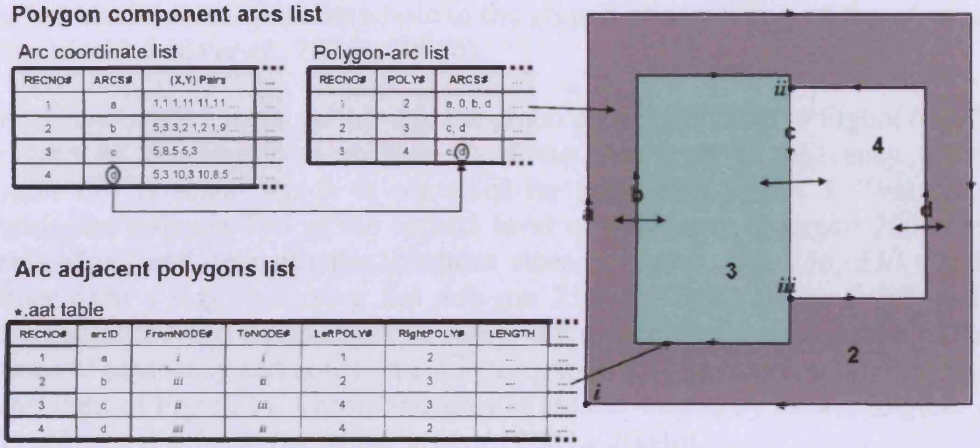


Figure 5. Combination of information to retrieve polygon adjacencies.

2.2.2 Graph representation

There are two common methods to represent graphs in a computer (Sedgewick, 1988, 1998, 2002): the *adjacency matrix* representation; and the *adjacency lists* representation. Both graph representations are arrays of simpler data structures, one for each vertex describing the edges incident on that particular vertex. The adjacency matrix is the simpler data structure and is implemented as an indexed two-dimensional array; the implementation of adjacency lists is based upon an array of linked lists. Because the graphs of adjacencies obtained in our case are fairly sparse, the adjacency lists representation appears to be the most appropriate data structure to use (Kruse *et al.* 1991, Schalkoff 1992, all cited in Barr and Barnsley, 1997; Sedgewick, 1988, 1998, 2002). It allows us to process all the edges of a graph in time proportional to $V+E$ (*i.e.* the total number of vertices plus the number of edges).

The processing was implemented in an application developed in C foreseeing the advantages and potentialities of pointer structures in C for graph analysis (Kelley *et al.*, 1990).

2.3 Bases for the analytical analysis method

2.3.1Preleminaries

In Figure 4 there is a representation of the graph of adjacencies for a particular area within the initial data set. Different levels of adjacency are indicated. Polygon *3*, highlighted on the bottom right, whose corresponding vertex is located right in the centre of the graph (*vd.* yellow circle), is the most connected flat polygon and the one with the longest perimeter. It corresponds indeed to the ground polygon and therefore constitutes the *useful external border* (Nardinocchi *et al.*, 2003), with which the graph drawing was started, and from where sequences of adjacencies/containments make most sense in terms of the urban scene.

It is possible to retrieve further geographical information by analysing different paths within the generated graph of adjacencies. Starting from the useful external border, a simple visual observation of the represented sequences of levels of adjacency between vertices along some graph paths, may tell us that, for instance, a vertex in the end of a path, representing the highest level of adjacency in that particular graph

path, is a candidate to be either a hole in the ground or something on top of an urban feature (de Almeida *et al.*, 2004a, 2004b).

To give an example, let us go through the graph path highlighted in Figure 6 (a detail of Figure 4). Starting from polygon *3*, at the first level of adjacency the steep polygon *198* is found which is contained by previous polygon *3*. That, in turn, contains flat polygon *200* at the second level of adjacency. Polygon *200* contains several others and, in particular, contains steep polygons *250*, *256*, *260* which all together form a ring *containing* flat polygon *257*, belonging to the fourth and last level of adjacency. In terms of urban scene, the meaning of this sequence of spatial relations of adjacency and containment is the existence of a building (pictured on the bottom right of Figure 6), whose boundary is almost shaped by the *rectangular* dark green polygon displayed (de Almeida *et al.*, 2004a, 2004b).

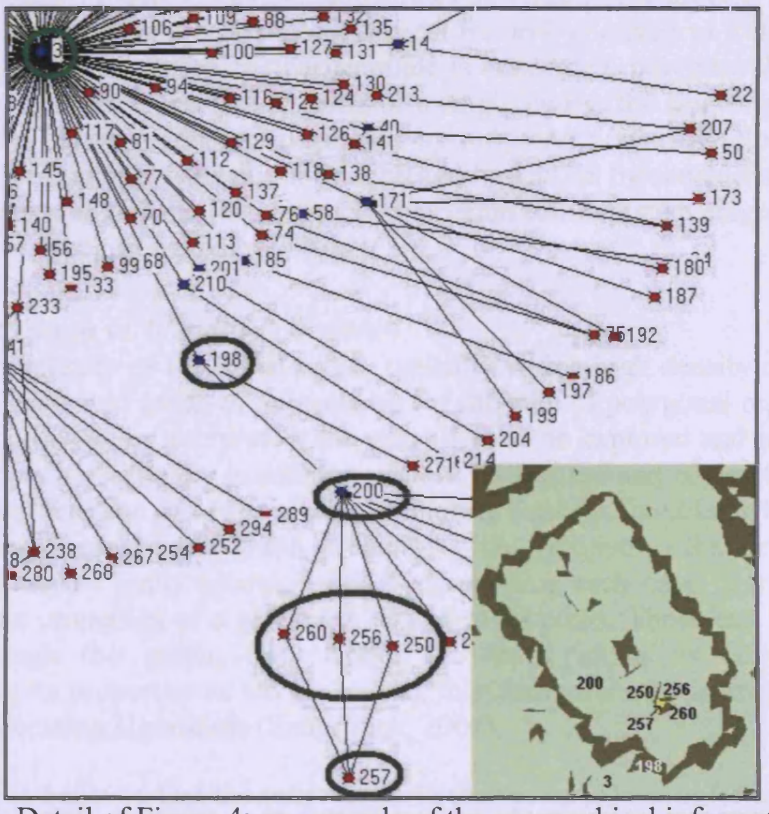


Figure 6. Detail of Figure 4: an example of the geographical information that may be inferred from a graph path in the context of the urban scene.

In the example given above polygons *250*, *256* and *260* are separate entities though they belong to the same class. This fact is mainly due to the GIS package being used; as explained in section 2.2.1, two polygons are considered adjacent if both share at least one arc (*vd.* Figure 7). What happens in this case is that polygon *250* meets polygon *256* at a node, *256* meets *260* at another node, and *260* in turn closes the ring meeting *250* at a third node. These polygons were not merged into one polygon because of the reasons described in section 2.1. Given this fact, there are no edges in the graph of adjacencies linking their corresponding vertices. Shown on top of the polygons in Figure 7 is the graph structure linking the vertices representing the polygons.

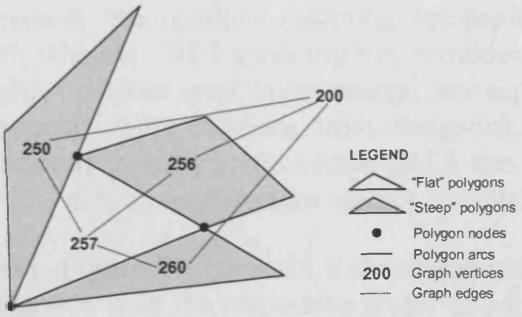


Figure 7. Detail of Figure 6: a special case of the spatial relation containment, between a ring of polygons and a single polygon.

This is a particular example of a situation which can be detected across the whole map of polygons. In reality, this fact constitutes an issue in the process of the graph analysis, for this particular case of containment has to be derived as it is not explicit in the graph. As far as this particular example is concerned, polygons *250*, *256* and *260* shall be considered component parts of a single entity (the ring of polygons) to be represented in the graph of adjacencies by only one vertex, as illustrated by a green ellipse in Figure 6. This might be achieved in ArcGIS by considering the three-polygon ring as a composite feature, such as a *region* in ESRI's coverage data model, comprising three polygonal components.

2.3.2 Depth-first search vs. breadth-first search

Given the complexity of the urban scene, typically with a high density of small size features, the generated graph of adjacencies for the map of polygonal regions is also complex. In order to be interpreted, the graph has to be explored and its properties determined by systematically examining each of its vertices and edges. Carrying out this task is cumbersome and equivalent to exploring a maze. Should one be interested in determining some simple graph properties, like computing the degrees of all vertices, this can be easily accomplished by examining each edge. But many other more complex properties of a graph are related to its paths. Those can be learnt by moving through the graph, from vertex to vertex along its edges, and by understanding its properties as we go. Indeed, this abstract model is used by most of the graph-processing algorithms (Sedgewick, 2002).

Therefore, it is believed that the retrieval of further geographical information, which was described above, constitutes that sort of analysis that is possible to carry out if based upon the graph traversal. In the literature two different algorithms are available to accomplish this task: the *depth-first search* (DFS) and the *breadth-first search* (BFS), (Sedgewick, 1988, 1998, 2002). Although both algorithms visit systematically all the graph nodes, the manner they operate is different. Briefly, depth-first search "moves from node to node, backing up to the previous node to try the next possibility whenever it has tried every possibility at a given node", it can be compared to "a single searcher probing unknown territory as deeply as possible" (Sedgewick, 1998). In contrast, breadth-first search "exhausts all the possibilities at one node before moving to the next", it amounts to "an army of searchers fanning out to cover the territory" (Sedgewick, 1998). From the implementation point of view, DFS can be either recursive or can use an explicit pushdown stack (in our application the recursive function implementation was used), whereas BFS uses FIFO (*first in, first out*) queues for its implementation.

As far as DFS is concerned, "the resulting spanning tree depicts the sequence of the traverse function calls"; whereas "BFS spanning tree provides a compact description of the dynamic properties of this level order search, corresponding one branch to each connected component" (both citations from Sedgwick, 2002). In both cases each tree vertex corresponds to each graph vertex and a tree edge corresponds to a traversed graph edge, thus non-traversed edges are not considered.

For illustration purposes, Figure 8 represents a simulated map of simple polygons, not classified in any manner, with the respective graph of adjacencies drawn on top of them. Let us choose polygon *2* as the root: the resulting spanning trees, both DFS and BFS, are pictured in Figure 9.

For the sake of flexibility, both graph-search algorithms, DFS and BFS, were implemented in our application in C, giving the user the choice of which algorithm to run. However, given the way the respective algorithms are conceived, it seems that the spanning tree resulting from the BFS traversal (broad and shallow, *vd.* Figure 9) is more meaningful in terms of the urban scene: if we look at its several short branches, these indeed appear to correspond to different urban features. In contrast, the DFS spanning tree (slim and deep, *vd.* Figure 9) does not seem to be as easily related to urban features as the previous one, for the interpretation of its long deep path does not appear to be straightforward.

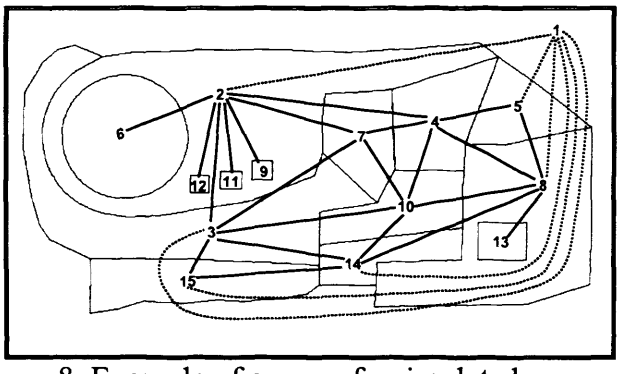


Figure 8. Example of a map of a simulated scene and respective graph of adjacencies.

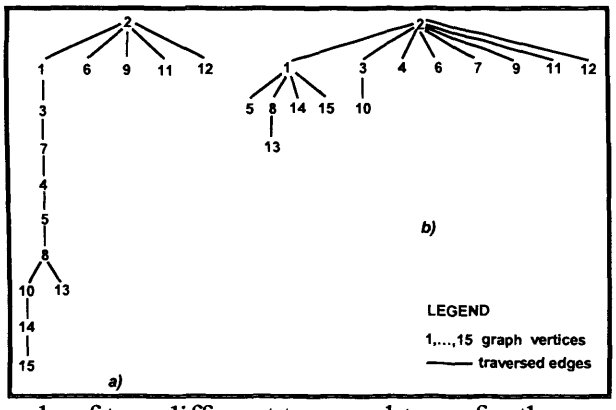


Figure 9. Example of two different traversal trees for the graph in Figure 8, both having the same root: *a)* depth-first search; *b)* breadth-first search.

Both DFS and BFS were implemented in such a way that, again for the sake of flexibility, it is possible to traverse the graph starting from any of its vertices. Nevertheless, we might be ultimately interested in considering this analysis starting

from the useful external border (ground polygon), from where the sequences of adjacency (which in some cases represents containment too) make most sense in terms of the scene.

Although it is possible at the moment to visualise either spanning tree as an interim result of analysis, the main aim being sought is to extend the analysis algorithm and eventually being able to visualise its results. The ongoing developments for this purpose are being based in particular on the BFS and comprise the implementation of further analytical rules; they take into account namely, the valence (or degree) of each vertex in the initial graph, and the level of adjacency of each vertex in the tree. For instance, while traversing the whole graph, the application counts the levels of adjacency and analyzes the depth of the tree; hence, it is possible to know how many levels of adjacency/containment a graph vertex is away from the root in the tree generated by a specific search.

3. Conclusions and further work

3.1 Conclusions

The use of graph theory is becoming an increasingly important tool for the analysis and understanding of complex urban scenes. Starting from initially unstructured geospatial data sets of urban areas (thus, no prior knowledge of the spatial entities is assumed), this paper shows how a graph-theoretic approach can be applied in these circumstances towards the analysis of urban scene spatial topology.

The theoretical and practical methods developed so far to analyze entities within a LiDAR data set of urban environment have been presented. In particular, the merits of depth-first and breadth-first search algorithms in analysing the structure of the urban spatial topology were discussed. Given the different ways both algorithms operate in traversing a graph, it was noted how BFS results are more meaningful in terms of the urban scene: the BFS tree branches are connected components of the original graph, and represent the shortest path between the root and their leaf (Sedgewick, 2002); it seems that they can be related to potential urban features.

Thus, the implementation of the graph analysis procedure is being based upon BFS. It traverses the graph looking for sequential relationships of containment amongst the sequences of adjacency: *containment-first search* (CFS). In fact, where containment occurs there is a high likelihood of an urban feature being present.

However, for an effective interpretation of the urban scene topology, developing CFS simply based on BFS is not sufficient. The CFS procedure has to be extended in order to be able to detect the spatial relation of containment in a broader sense (*vd.* 2.2.1). Indeed, there are some particular cases of containment which are not explicit in the graph (*vd.* example in Figure 7). In order to address this issue, we propose that the spatial relation of *touching* between steep polygons should be taken into consideration, so as these particular cases can be derived by defining polygon-ring containments. Further investigation of this aspect will be object of future work.

3.2 Further work

Currently, the system is being extended to the visualisation of graphs, and more importantly of the resulting traversal trees. A visual representation of the urban

topological analysis is also under consideration. In fact, the human brain is sensitive to the visual representation of real scenes, and visual analysis can often reveal patterns not discernable by current automated analysis techniques.

Thus, it reveals relevant the incorporation of capabilities for visual representation of both, the urban scene topology and its analysis, in the application under development. In particular, a graph traversal tree is a simpler representation of the initial graph that is well worthy of careful study (Sedgwick, 2002). The interesting aspect in visualising a traversal tree is that it contains almost the same information as the initial graph, but it is displayed in a slightly different way, making the graph structure somewhat more explicit. Given the dimension and complexity of the original graphs of adjacencies, we believe that the observation of the traversal trees is useful in detecting the existence of urban structures.

Also, the possibility of linking up the graph analysis application with the GIS environment is being investigated. In fact, it is believed that the utility of the visual representation of the topological data structures described in section 2.3.2 should be enhanced in terms of scene analysis if the visualisation tool is coupled with the original map. The ultimate goal is the implementation of functionalities to display dynamically the initial map of polygonal regions according to the results of the urban topology analysis.

For this purpose, we are currently working on the development of an interactive tool. This is being implemented in ArcMap (ArcGIS 8.3) using its embedded programming language, Visual Basic for Applications (VBA). Developing the application in this original environment has the advantage of being able to perform graph analysis based upon some of the polygon attributes, which can be withdrawn from the respective Polygon Attribute Table (PAT). In turn, this table can also be updated to integrate numerical results of the analyses carried out.

At the moment, the user can carry out any particular visual inspection, say in the original map of polygons, and simultaneously being able to obtain the respective traversal tree starting from the chosen root (*vd.* Figure 10). Other functionalities, like accessing directly a polygon's attributes when its respective vertex is selected in the spanning tree, or when selecting a vertex in the tree the corresponding polygon being highlighted on the map, are also being implemented.

Furthermore, the interactive capabilities should be useful in dealing with complex scenes, like the existence of a discontinuous ground polygon. In this situation, the corresponding graph of adjacencies will consist of different sub-graphs connected to each other by single linking edges. In such an interactive tool, the capability of analyzing which sub-graph corresponds to which area on the map, and simultaneously obtaining the respective traversal tree, appears to be interesting.

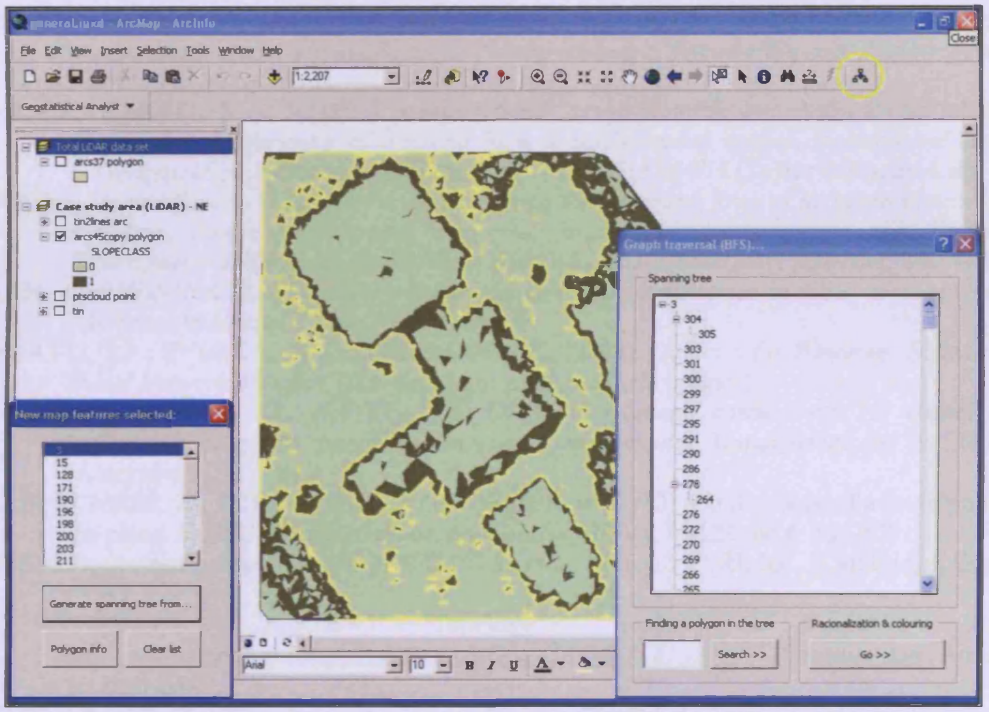


Figure 10. An interactive tool for the visualisation of topological data structures generated from the analyses of the adjacency graph. (Root – polygon *3*, selected in yellow)

Further work will also entail the implementation of other possible rules to enable the analysis process explained in section 2.3, eventually leading to the aggregation of graph vertices into identified meaningful structures. These, in turn, should be clustered into homogenous regions (Forberg & Raheja, 2002). After the delineation of cluster shapes, an analysis process will have to be accomplished, either by pattern recognition or interpretation procedures (Toussaint, 1980b). The aim of the ultimate cluster shapes analysis is the retrieval of higher-level information, *e.g.* sets of buildings, vegetation areas, and say land-use parcels.

We note that this application is at the same time to investigate rules for urban scene analysis and graphic representation of results. We expect the resulting system to be useful to support land-use mapping, image understanding or, in more general terms, to support clustering analysis and cartographic generalisation processes.

References

de ALMEIDA J.-P., MORLEY J. G., DOWMAN I. J. (2004a). A graph-based scene analysis technique. *Proceedings of GISRUK 2004*, 365-369 (Norwich, UK).

de ALMEIDA J.-P., MORLEY J. G., DOWMAN I. J. (2004b). A graph-based approach for higher order GIS topological analysis. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXV, Part B4, Commission IV*, 1024-1028 (Istanbul, Turkey: XX Conference of ISPRS).

ANDERS, K.-H.; SESTER, M.; FRITSCH, D. (1999). Analysis of settlement structures by graph-based clustering. *Semantische Modellierung, SMATI 99*, 41-49 (Munich, Germany).

BARNSLEY, M.J.; BARR, S.L. (1998). Distinguishing urban land-use categories in fine spatial resolution land-cover data using a graph-based, structural pattern recognition system. *Comput., Environ. And Urban Systems*, Vol. 21, No. 3/4: 209-225 (Elsevier Science Ltd).

BARR, S.L., BARNSLEY, M.J. (1996). Inferring urban land-use from satellite sensor images using kernel-based spatial reclassification. *Photogrammetric Engineering and Remote Sensing*, No. 62: 949-958.

BARR, S.L., BARNSLEY, M.J. (1997). A region-based, graph-theoretic data model for the inference of second-order thematic information from remotely-sensed images. *International Journal of Geographical Information Science*, Vol. 11, No. 6: 555-576 (Taylor & Francis Ltd).

BARR, S.L., BARNSLEY, M.J. (2004). Characterising the structural form of an urban system using built-form connectivity model concepts. In *Spatial Modelling of the Terrestrial Environment* (Kelly, R.E.; Drake N.A.; Barr, S.L., eds.): 201-226 (Chichester, UK: Wiley).

BAUER, T.; STEINNOCHER K. (2001). Per-parcel land use classification in urban areas applying a rule-based technique. *GeoBIT/GIS 6*: 24-27.

BORGATTI, S.P.; EVERETT, M.G.; FREEMAN, L.C. (2002). *Ucinet 6 for Windows: Software for Social Network Analysis*. (Harvard, MA: Analytic Technologies).

BUNN, A.G.; URBAN, D.L.; KEITT, T.H. (2000). Landscape connectivity: A conservation application of graph theory. *Journal of Environmental Management*, no.59, 265-275 (Academic Press, USA).

EDELSBRUNNER, H.; KIRKPATRICK, D.G.; SEIDEL, R. (1983). On the shape of a set of points in the plane. *In IEEE Transactions on Information Theory*, Vol.29, no.4: 551-559.

ESRI (1995). *Understanding GIS, The ARC/INFO Method*, Lesson2. ESRI, Inc., (Cambridge, England - UK).

ESRI (2004). ArcGIS Topology [online]. Available from http://www.esriuk.com/products/support/ArcGIS%20UK%20EXT/Topology.asp [Accessed 10 June 2005].

EYTON, J.R. (1993). Urban land-use classification and modelling using cover-type frequencies. *Applied Geography*, Vol. 13: 111-121.

FORBERG, A.; RAHEJA, J.L. (2002). Generalization of 3D settlement structures on scale-space and structures recognition. Work Report, Institut für Photogrammetrie und Fenerkundung (Universität der Bundeswehr München), Lehrstuhl für Kartographie (Munich, Germany: Technische Universität München).

GIBBONS, A. (1989). *Algorithmic Graph Theory*. (Cambridge, UK: Cambridge University Press).

GROSS, J.; YELLEN, J. (1999). *Graph Theory and Its Applications* (Boca Ratón-Florida, USA: CRC Press).

KELLEY, A.; POHL, I. (1990). *A Book on C - Programming in C*. 2nd ed (Redwood City – CA, USA: Benjamin Cummings).

KIM, T.; MULLER, J.-P. (1999). Development of a graph-based approach for building detection. *Image and Vision Computing*, 17: 3-14 (Elsevier Science B.V.)

KIRKPATRICK, D.G.; RADKE, J.D. (1985). A framework for computational morphology. *Computational Geometry* (TOUSSAINT, G.T., ed.), 217-248 (Amsterdam, The Netherlands: North- Holland).

KRUSE, R.L.; LEUNG, B.P.; TONDO, C.L. (1991). *Data Structures and Program Design in C* (Prince Hall, New York - USA).

LAURINI, R.; THOMPSON, D. (1992). *Fundamentals of Spatial Information, 5* (London, UK: Academic Press).

NARDINOCCHI, C., GIANFRANCO, F., ZINGARETTI, P. (2003). Classification and filtering of laser data. ISPRS Workshop on *3D reconstruction from airborne laser scanner and InSAR data* (Dresden, Germany).

O'SULLIVAN, D.; TURNER, A. (2001). Visibility graphs and landscape visibility analysis. *International Journal of Geographical Information Science*, Vol.15, No.3: 221-237 (Taylor & Francis Ltd).

REED, C. (1999). GIS Users Shouldn't Forget About Topology [online]. In *GeoWorld Readers' Forum*. Available from http://www.geoplace.com/gw/1999/0499/499form.asp [Accessed 29 January 2004].

ROBERTS, S.A.; HALL, G.B.; CALAMAI, P.H. (2000). Analysing forest fragmentation using spatial autocorrelation, graphs and GIS. *International Journal of Geographical Information Science*, Vol.14, No.2: 185-204 (Taylor & Francis Ltd).

SCHALKOFF, R. J. (1992). *Pattern Recognition: Statistical, Structural and Neural Approaches.* (New York, USA: John Wiley).

SEDGEWICK, R. (1988). *Algorithms*. 2nd ed (Reading-MA, USA: Addison-Wesley).

SEDGEWICK, R. (1998). *Algorithms in C, Parts 1-4*. 3rd ed. Chap. 3, 4, 5 (Reading-MA, USA: Addison-Wesley).

SEDGEWICK, R. (2002). *Algorithms in C, Part 5*. 3rd ed. Chap. 17, 18 (Reading-MA, USA: Addison-Wesley).

STEEL, A.M.; BARNSLEY, M.J.; BARR, S.L. (2003). Inferring urban land use through analysis of the spatial composition of buildings identified in LiDAR and multispectral image data. *Remotely-Sensed Cities* (MESEV, T.V., ed.), (London, UK: Taylor & Francis Ltd).

TEMPERLEY, H. (1981). *Graph Theory and Applications* (Chichester, UK: Ellis Horwood Ltd).

THEOBALD, D.M. (2001). Topology revisited: representing spatial relations. *International Journal of Geographical Information Science*, Vol. 15, No. 8: 689-705 (Taylor & Francis Ltd).

TOUSSAINT, G.T., 1980a. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, Vol.12, No.4: 261-268. (Pergamon Press Ltd).

TOUSSAINT, G.T., 1980b. Pattern recognition and geometrical complexity. Proceedings of *V International Conference on Pattern Recognition*: 1324-1347. (Miami, USA).

URQUHART, R., 1982. Graph theoretical clustering based on limited neighbourhood sets. *Pattern Recognition*, Vol.15, No.3: 173-187. (Pergamon Press Ltd).

WILSON, R, 1996. *Introduction to Graph Theory* (4th ed.), (Harlow, UK: Longman).

K CD ROM

This thesis contains a CD ROM attached. Its contents are:

- A folder with the AML routines;
- A folder with the application in C implemented in a Microsoft Visual C++ workspace;
- A folder with the interactive tool for the visualisation of the urban topology, implemented in an ArcMap project.

L Other types of graphs

Common families

Trivial graphs

A *trivial graph* is a graph consisting of only one vertex and no edges (Gross and Yellen, 1999, pg.2).

Complete graphs, Regular graphs

A graph for which every pair of distinct vertices defines an edge is called a *complete graph* (Gibbons, 1989, pg.3; Wilson, 1996, pg.17; Gross and Yellen, 1999, pg.10). The complete graph with n vertices is denoted by K_n. In a *regular graph*, every vertex has the same degree; if this is k, then the graph is called k-regular (notice that K_n is (n-1)-regular) (Wilson, 1996, pg.18; Gross and Yellen, 1999, pg.12).

Platonic graphs

Of interest among the regular graphs are the *platonic graphs*, formed from the vertices and edges of the five regular platonic solids: the tetrahedron, octahedron, cube, icosahedron and dodecahedron (Wilson, 1996, pg.18; Gross and Yellen, 1999, pg.12).

Complete bipartite graphs

A *complete bipartite graph* is a bipartite graph in which each vertex in one of the bipartition subset, U, is joined to each vertex in the other bipartition subset, W. Any complete bipartite graph that has m vertices in one of its bipartition subsets and n in the other is denoted $K_{m,n}$ (Wilson, 1996, pg.18; Gross and Yellen, 1999, pg.11).

Bouquets and Dipoles

One-vertex and two-vertex non-simple graphs often serve, for instance, as building blocks for various interconnection networks, including certain parallel architectures. A graph consisting of a single vertex and n self-loops is called a *bouquet* and is denoted B_n. A graph consisting of two vertices and n edges joining them is called a *dipole* and is denoted D_n (Gross and Yellen, 1999, pg.13).

Eulerian and Hamiltonian graphs

Graphs containing walks that include every edge exactly once, ending at the initial vertex, are called *Eulerian graphs* (Gibbons, 1989, pg.153; Wilson, 1996, pg.31; Gross and Yellen, 1999, pg.34). The term *"Eulerian"* is in honour of the eminent Swiss mathematician Leonhard Euler (Gibbons, 1989, pg.185; Bollobás, 1998, pg.16). Eulerian graphs have a vertex-based analogue first studied by the British mathematician William R. Hamilton (*circa* 1856): graphs containing walks that include every vertex exactly once, ending at the initial vertex, are called *Hamiltonian graphs* (Gibbons, 1989, pg.169; Wilson, 1996, pg.35; Bollobás, 1998, pg.14; Gross and Yellen, 1999, pg.41). Notice that in the example in Figure 3.2 to Figure 3.4, the graph is a Hamiltonian one since it is possible to build up a walk including all the vertices once. A suitable walk is $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_1$.

Graph modelling applications

Different kinds of graphs can emerge in modelling real-world situations. Sometimes, simple graphs are adequate; at other times, non-simple graphs are needed.

In modelling some of those specific situations, graph elements, vertices and edges, may have additional attributes, such as colour or weight, or anything else useful to a particular purpose. Graph models can be classified into different categories according to their characteristics. For instance, the network of one-way streets of a city requires a model in which each edge is assigned a direction; a computer programme is likely to have loop structures. These two examples require graphs respectively with

directions on their edges, or with connections from a vertex to itself (Gross and Yellen, 1999).

In the past, these different types of graphs were regarded as separate entities, each one with its own set of definitions and properties. By considering all of these various graphs at once, Gross and Yellen (1999) introduced a new conception: a unified approach.

Multi-graphs

Considering again the sample in Figure 3.1, let us suppose that a new extra road is built to join points v_4 and v_2. To represent this new situation, it is necessary to replace edge e_6 by what is defined in section 3.2 as a multi-edge, and therefore the new graph obtained is called a *multi-graph* (Wilson, 1996, pg.3; Gross and Yellen, 1999, pg.2; Roberts *et al.*, 2000), like the one represented in Figure L.1.

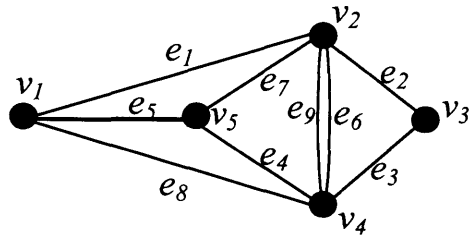


Figure L.1 – A general example of a multi-graph. (After Wilson, 1996)

Directed graphs

Supposing our road sample represents a set of one-way streets, the directions of the one-way streets should be indicated by arrows, as in Figure L.2. This kind of situation introduces the study of so called *directed graphs* (or *digraphs*), in which vertices are joined by directed edges (Wilson, 1996, Chap.7; Gross and Yellen, 1999, pg.2; Roberts *et al.*, 2000).

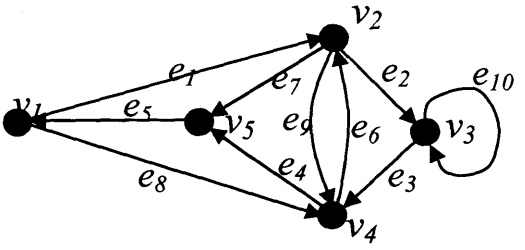


Figure L.2 – A general example of a digraph. (After Wilson, 1996)

Let us add now to the sample digraph a loop. Supposing that a loop road is built at v_3, this situation is indicated by drawing an edge, e_{10}, from v_3 to itself. As pointed out previously in section 3.2, the digraph adjacency matrix, denoted by A, is not necessarily symmetric. For a digraph with n vertices, A is the nxn matrix given by (Gross and Yellen, 1999, pg.77):

$$A = [a_{ij}],\ \text{where } a_{ij} = \begin{cases} \text{the number of arcs from } v_i \text{ to } v_j, \text{ if } v_i \neq v_j \text{ or} \\ \text{the number of self-loops at } v_j, \text{ if } v_i = v_j; \\ 0 \quad \text{otherwise} \end{cases} . \qquad \text{(Equation L.1)}$$

Thus, the respective adjacency matrix for the example in Figure L.2 is:

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} . \qquad \text{(Equation L.2)}$$

The incidence matrix of a digraph is analogously defined as the matrix whose rows and columns are indexed by some orderings of $V(G)$ and $E(G)$, respectively, such that (Gross and Yellen, 1999, pg.75):

$$X = [x_{ij}],\ \text{where } x_{ij} = \begin{cases} 2 & \text{if edge } j \text{ is a self - loop at vertex } i, \\ 1 & \text{if vertex } i \text{ is the head of the edge } j, \\ -1 & \text{if vertex } i \text{ is the tail of the edge } j, \\ 0 & \text{otherwise} \end{cases} . \qquad \text{(Equation L.3)}$$

And therefore, the incidence matrix for the example shown in Figure L.2 is:

$$X = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} \end{array} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & -1 & 0 & -1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array} . \qquad \textit{(Equation L.4)}$$

To give a particular example of the application of these concepts, recall the ESRI's ArcInfo coverage data model.

Random graphs

Additional attributes may be attached to graph elements. Suppose that for purposes of transportation of certain goods it is useful to know the capacity of each channel. A number next to the respective edge, representing the maximum amount that can pass through that channel, can indicate this. But in other situations, supposing that particular vertices and edges are present, or not, in the graph following a probability distribution, which is derived either empirically or theoretically, the attachment to those graph elements of their respective probability values might be useful (*vd.* edge values in Figure L.3). In these situations, graphs are called *random graphs* (Roberts *et al.*, 2000; Bollobás, 2001).

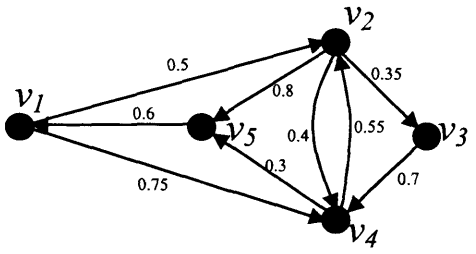


Figure L.3 - A general example of a random graph.
(After Wilson, 1996)

Hypergraphs

Hypergraphs are graphs that also include sets of vertices as another structural element independent of edge interconnections (Bollobás, 1998, pg.7; Roberts *et al.*, 2000).

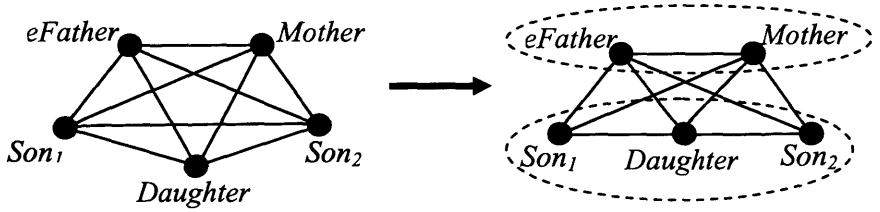


Figure L.4 - A general example of a hypergraph.

As an example, let us consider "family" as a relation connecting two or more people (*vd.* Figure L.4). If each person is a vertex, a family edge connects the father, mother, and all of their children; so, G = *(people, family links)* is a graph representing the "family". It should be borne in mind that each edge can represent a different kind of link, and therefore the set "family links" can be split up into two or more subsets of different kinds of family links depending upon the binary relations: "married to", which connects a man and a woman; or "child of", which is directed from a child to his or her father or mother; or "sibling of", which connects brothers and sisters. In this example, two sets of vertices can be considered within the graph: "the parents" and "the children", who are both represented in the figure by the dotted lines. It is this grouping that means that a graph is a hypergraph.